



University  
of Glasgow

Chiew, Thiam Kian (2009) *Web page performance analysis*.  
PhD thesis.

<http://theses.gla.ac.uk/658/>

Copyright and moral rights for this thesis are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the Author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the Author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

# **WEB PAGE PERFORMANCE ANALYSIS**

by

**THIAM KIAN CHIEW**

Department of Computing Science

This thesis is submitted to the  
Faculty of Information and Mathematical Sciences  
at the University of Glasgow  
for the degree of Doctor of Philosophy

**March 2009**

## DECLARATION

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree. This thesis is the result of my own investigations. Resources referenced are acknowledged.

Signed :  \_\_\_\_\_

Name : THIAM KIAN CHIEW

Date : 24 MARCH 2009

# ABSTRACT

Computer systems play an increasingly crucial and ubiquitous role in human endeavour by carrying out or facilitating tasks and providing information and services. How much work these systems can accomplish, within a certain amount of time, using a certain amount of resources, characterises the systems' performance, which is a major concern when the systems are planned, designed, implemented, deployed, and evolve. As one of the most popular computer systems, the Web is inevitably scrutinised in terms of performance analysis that deals with its speed, capacity, resource utilisation, and availability. Performance analyses for the Web are normally done from the perspective of the Web servers and the underlying network (the Internet). This research, on the other hand, approaches Web performance analysis from the perspective of Web *pages*. The performance metric of interest here is response time. Response time is studied as an attribute of Web *pages*, instead of being considered purely a result of network and server conditions. A framework that consists of measurement, modelling, and monitoring (3Ms) of Web pages that revolves around response time is adopted to support the performance analysis activity. The *measurement* module enables Web page response time to be measured and is used to support the *modelling* module, which in turn provides references for the *monitoring* module. The monitoring module estimates response time. The three modules are used in the software development lifecycle to ensure that developed Web pages deliver at worst satisfactory response time (within a maximum acceptable time), or preferably much better response time, thereby maximising the efficiency of the pages. The framework proposes a systematic way to understand response time as it is related to specific characteristics of Web pages and explains how individual Web page response time can be examined and improved.

# TABLE OF CONTENTS

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Table of Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Acronyms</b>	<b>xiii</b>
<b>Acknowledgement</b>	<b>xvi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Hypothesis and Thesis Statement	3
1.2 Thesis Structure	4
<b>2 Background</b>	<b>7</b>
2.1 The Internet	7
2.1.1 Overview and the History	7
2.1.2 Packet Switching and Internet Protocol Suites	17
2.2 The World Wide Web	22
2.3 Performance	28
2.3.1 Performance Metrics	28
2.3.2 Performance Benchmarking	30
2.3.3 Performance Analysis	31
2.3.3.1 Categories and Issues	31
2.3.3.2 Goals and Steps	33
2.3.4 Performance and Web-based Systems	33
2.4 Response Time	35
2.5 Improving Response Time	38

2.5.1	Caching	39
2.5.2	Prefetching	41
2.5.3	Content Distribution Network	43
2.5.4	Load Balancing	44
2.5.5	Decomposition of Dynamic Web Content	47
2.5.6	Server Tuning	49
2.5.7	Customisation	50
2.5.8	Web Standards	51
2.5.9	Web Technologies (Software)	52
2.5.10	Web Page Content: Composition, Generation and Structure	54
2.5.11	Hardware	57
2.6	Measuring Response Time	59
2.6.1	Active Probing	59
2.6.2	Server-side Measurements	60
2.6.3	Client-side Measurements	64
2.7	Conclusion	66
<b>3</b>	<b>Problem Statement</b>	<b>68</b>
3.1	Background and the Problems	68
3.2	Research Framework and Approach	70
3.3	Conclusion	74
<b>4</b>	<b>Measuring and Decomposing Web Page Response Time</b>	<b>75</b>
4.1	Components of Web Page Response Time	75
4.2	The First Method: Proxy Server	77
4.2.1	Muffin HTTP Proxy Server	84
4.2.2	Advantages and Disadvantages	85
4.3	The Second Method: Web Page with Frames	86

4.3.1	Advantages and Disadvantages	89
4.4	Measurement and Decomposition of Response Time	90
4.4.1	The Dynamic JSP Home Page	91
4.4.2	The Static HTML Home Page	92
4.5	Discussion	93
4.5.1	First Access to the JSP Home Page	93
4.5.2	Caching	96
4.5.3	Dynamicity of Web Page	98
4.5.4	DNS Lookup	99
4.5.5	Rendering and Correlations between Different Time Components	100
4.6	Conclusion	102
<b>5</b>	<b>Modelling Web Page Response Time</b>	<b>103</b>
5.1	Modelling	103
5.2	Web Page Complexity and Response Time	105
5.3	Creation Complexity	109
5.3.1	Static and Dynamic Pages	110
5.3.2	State	113
5.3.3	Modelling Creation Complexity	114
5.4	Content Complexity	116
5.5	Applying and Validating the Models	117
5.5.1	Lines of Code	118
5.5.2	Number of Embedded Objects	120
5.5.3	Total Size of the Page	123
5.5.4	JavaScript and CSS	125
5.5.5	Programming Constructs	127

5.5.6	Database Access	130
5.5.7	Cookies	133
5.6	Conclusion	135
<b>6</b>	<b>Monitoring Web Page Response Time</b>	<b>137</b>
6.1	Monitoring	137
6.2	Server Access Log	139
6.2.1	Web Log Analysis	143
6.3	Estimating Web Page Response Time Based on Server Log Analysis	144
6.4	Prototype Monitoring System	151
6.5	Testing the Prototype Monitoring System	155
6.6	Conclusion	157
<b>7</b>	<b>Making Use of Measurement, Modelling and Monitoring To Enhance Web Page Performance: An Example</b>	<b>159</b>
7.1	Scenario 1: Designing Web Pages with Desired Response Time	159
7.2	Scenario 2: Real Time Monitoring of Web Page Response Time	166
7.3	A Supporting Tool for Examining Database Queries in a Web Page	169
7.3.1	Developing the Tool	170
7.3.2	Testing the Tool	174
7.4	Conclusion	179
<b>8</b>	<b>Conclusion</b>	<b>180</b>
8.1	Overview	180
8.1.1	Context	180
8.1.2	Scope	181
8.1.3	Focus	182
8.2	Contributions	182
8.2.1	Measurement (Chapter 4)	182



8.2.2	Modelling (Chapter 5)	183
8.2.3	Monitoring (Chapter 6)	183
8.2.4	Application of the 3Ms (Chapter 7)	184
8.3	Review of the Thesis Statement	186
8.4	Limitations	187
8.5	Future Work	187
8.5.1	Improvement of the 3Ms	188
8.5.1.1	Measurement	188
8.5.1.2	Modelling	188
8.5.1.3	Monitoring	188
8.5.1.4	Supporting Tools for the 3Ms	189
8.5.2	Incorporating Response Time as Part of the Web Page Design Process	189
8.5.3	The 3Ms and Web Page Maintenance	189
<b>References</b>		<b>192</b>
<b>Bibliography</b>		<b>201</b>
<b>Appendices</b>		<b>208</b>
1	Notations of Times	209
2	Parsing of HTTP Requests and Responses by the Muffin Proxy Server	210
3	Response Time Measurement Filter for Muffin Proxy Server: The Source Code	212
4	Response Time and Responsiveness Monitor: The Source Code	214
5	Database Queries Examining Tool: The Source Code	252
6	ColdFusion Files Used to Test the Database Queries Examining Tool	265

## LIST OF FIGURES

2.1(a)	Data Flows in a Circuit Switched Network	9
2.1(b)	Data Flows in a Packet Switched Network	9
2.2	Three-tier LAN (Stallings, 2005)	11
2.3	Hierarchical Structure of Domain Name System	13
2.4	Comparison between TCP/IP Protocol Suite and the OSI Model (Stallings, 2005)	20
2.5	Protocol Data Units (PDUs) in the TCP/IP Architecture (Stallings, 2005)	20
2.6	TCP/IP Concepts (Stallings, 2005)	22
2.7	HTTP Request-response Sequences for Downloading a Page	27
2.8	Throughput and Response Time as Functions of Work Load (Kotsis, 2006)	30
2.9	Load Balancing using DNS Server and Connection Routers	46
2.10	A Dynamic Page Viewed as a Quasi-static Document Template Filled with Objects Exhibit Different Cacheability Characteristics (Shi <i>et al.</i> , 2003)	48
2.11	Active Probing: Agents Distributed across the World Actively Probing the Server to Perform Measurements	60
2.12	Using Server Log Analysis to Estimate Response Time	63
2.13	EtE Monitor Architecture (Cherkasova <i>et al.</i> , 2003)	64
2.14	Client Side Measurement of Response Time using JavaScript Instrumented Links	65
3.1	Research Framework	72
4.1	Three Parties Involved in Retrieving a Web Page	75
4.2	Measuring and Decomposing Response Time Using Muffin Proxy Server	77
4.3	JavaScript Code to Measure and Display Page Rendering Time	78

4.4	Communications between the Client and Server for Downloading a Web Page with One Embedded Object	80
4.5	Communications between the Client and Server with Overlapping Transmission and Server Processing Times	81
4.6	Web Page with Two Frames for Measuring Response Time	87
4.7	JavaScript Code for the Page-with-Frames Method	89
4.8	Online Bookstore Home Page	91
5.1	Four levels of Thinking Hierarchy	104
5.2	Causal Link between Web Page Complexity and Response Time	107
5.3	Distribution of Times Measured for Web Pages to Compare the Effect of LOC	119
5.4	Response Time as a Function of Number of Embedded Objects	122
5.5	Distribution of Times Measured for Web Pages to Compare the Effect of Number of Embedded Objects	123
5.6	Response Time as a Function of Total Page Size	124
5.7	Distribution of Times Measured for Web Pages to Compare the Effect of Total Page Size	125
5.8	Distribution of Times Measured for Web Pages to Examine the Effect of JavaScript and CSS	127
5.9	Distribution of Times Measured for Web Pages to Compare the Effect of Programming Constructs	129
5.10	Distribution of Times Measured for Web Pages to Compare the Effect of Database Access	132
5.11	Distribution of Times Measured for Web Pages to Examine the Effect of Cookies	134
6.1	Excerpt of a Server Access Log	145
6.2	Conceptual Model of the Monitoring System	151
6.3	Reports for the Monitoring System	153
6.4	System Start-up	153
6.5	Selecting Server Access Log to be Analysed	153

6.6	Overall Report for the Monitoring	154
6.7	Detailed Report for a Selected Page	154
6.8	Comparison between Measured and Monitored Results for Page 1	156
6.9	Comparison between Measured and Monitored Results for Page 2	156
6.10	Comparison between Measured and Monitored Results for Page 3	156
7.1	Flowchart Depicting the Process of Applying Measurement and Modelling Modules to Web Page Design	160
7.2	Steps for Web Page Response Time Analysis	163
7.3	Applying Content Complexity Models for Web Page Response Time Analysis	164
7.4	Applying Creation Complexity Models for Web Page Response Time Analysis	165
7.5	Real Time Monitoring of Web Page Response Time	166
7.6	SQL Select Statements' Syntax Expressed in the EBNF	172
7.7	Report for Database Queries Analysis for Response Time Improvement	173
7.8	Proposed Interface for Comparison between Web Pages on the Usage of Database Queries	178
7.9	Proposed Interface for Detailed Information on the Usage of Database Queries in a Web Page	178
8.1	Modularised Component-based View of a Web Page	190
A2.1	Parsing of HTTP Requests by Muffin Proxy Server	210
A2.2	Parsing of HTTP Responses by Muffin Proxy Server	211

## LIST OF TABLES

2.1	Comparison among Performance Analysis Techniques	32
2.2	Peak Rate Supported by Apache 1.3.27 Web Server in Producing Different Types of Responses	55
4.1	First and Subsequent Accesses to the JSP Home Page in Milliseconds	93
4.2	Subsequent Accesses to the JSP Home Page with and without Caching in Milliseconds	96
4.3	Accesses to the HTML Home Page with and without Caching in Milliseconds	97
4.4	Accesses to the JSP and HTML Home Pages without Caching in Milliseconds	98
4.5	Accesses to the HTML Home Page without Caching Using URL and IP Address in Milliseconds	99
4.6	Correlation Coefficients between $T_t$ , $T_d$ , and $T_g$	100
4.7	Correlation Coefficients between $T_{sp}$ and $T_r$ , $T_d$ , and $T_g$	101
5.1	Details for Pages Differ in LOC	118
5.2	Times Measured for Web Pages to Compare the Effect of LOC	119
5.3	Details for Pages Differ in Number of Embedded Objects	121
5.4	Times Measured for Web Pages to Compare the Effect of Number of Embedded Objects	121
5.5	Details for Pages Differ in Total Page Size	123
5.6	Times Measured for Web Pages to Compare the Effect of Total Page Size	124
5.7	Details for Pages that Include Elements	126
5.8	Times Measured for Web Pages to Examine the Effect of JavaScript and CSS	126
5.9	Details for Pages with Different Programming Construct Complexity	128
5.10	Times Measured for Web Pages to Compare the Effect of Programming Constructs	128
5.11	Details of Programming Constructs in Pages 4a, 4b, and 4c	130

5.12	Details for Pages with Different Database Access Attributes	131
5.13	Times Measured for Web Pages to Compare the Effect of Database Access	131
5.14	Details for Pages to Examine the Effect of Cookies on Response Time	133
5.15	Times Measured for Web Pages to Compare the Effect of Cookies	134
6.1	Common Logfile Format Information Fields	141
6.2	Comparison between Measured and Monitored Results	155
7.1	Web Page Response Time and Responsiveness Monitoring Table	167
7.2	Comparison between Automated and Manual Inspections of Web Pages	175
A1	Notations of Times Used in this Thesis	209

## LIST OF ACRONYMS

AARNet	Australian Academic and Research Network
Ajax	Asynchronous JavaScript and XML
AOP	Aspect-oriented Programming
ARPA	Advanced Research Projects Agency
AS	Autonomous System
ASP	Active Server Pages
BBS	Bulletin Board System
CDN	Content Distribution Network
CERN	European Organization for Nuclear Research
CFML	ColdFusion Markup Language
CGI	Common Gateway Interface
CLF	Common Logfile Format
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DNS	Domain Name System
EBNF	Extended Backus-Naur Form
EJB	Enterprise Java Beans
FDDI	Fiber Distributed Data Interface
HCI	Human-Computer Interaction
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
INRIA	French National Institute for Research in Computer Science and Control

IP	Internet Protocol
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union's Telecommunication Standardization Sector
JSP	Java Server Pages
JVM	Java Virtual Machine
LAN	Local Area Network
LOC	Lines of Code
MTBF	Meantime Between Failure
MTBR	Meantime Between Repair
NCP	Network Control Program
NCSA	National Center for Supercomputing Applications
NIC	Network Interface Card
NORSAR	Norwegian Seismic Array
NSF	National Science Foundation
NSFNET	National Science Foundation Network
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PHP	Hypertext Preprocessor
QoS	Quality of Service
RAM	Random Access Memory
SPEC	Standard Performance Evaluation Corporation
SQL	Structured Query Language
SRI	Stanford Research Institute
SUT	System Under Test
TCP	Transmission Control Protocol



TLD	Top Level Domain
TPC	Transaction Processing Performance Council
TTL	Time-to-Live
TWT	Tolerable Waiting Time
UCLA	University of California, Los Angeles
UCSB	University of California, Santa Barbara
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
US	United States of America
UUCP	Unix to Unix CoPy
W3C	World Wide Web Consortium
WAN	Wide Area Network
WSC	Web Stream Customizer

## ACKNOWLEDGEMENT

This thesis hasn't been produced with my sole work. Many remarkable individuals helped me out in various forms since my research started in February 2004 until its completion.

First and foremost, I wish to express my greatest gratitude to my supervisor, Dr Karen Renaud. She has guided me through the paths and pitfalls of research in general and computing science specifically. Her invaluable advice has widened my knowledge and improved my skills in conducting research. She provided me with a sufficient level of freedom to explore my own ideas yet enough guidance to ensure that I was always on the right track in pursuing my PhD degree.

Many members of the Department of Computing Science at the University of Glasgow also gave me helpful consultations and shared their ideas with me, which has made this thesis a better piece of work. They are Dr. Richard Cooper (my second supervisor), Prof. Ray Welland, Dr. Mohamed Ould-Khaoua, Dr. Lewis McKenzie, Dr. Matthew Chalmers, Dr. Marek Bell, Miss Dora Galvezcruz, Miss Mariam Alawadi, Mr. Mohamed Al Fairuz, and many others to whom I owe the same amount of appreciation even though their names are not mentioned here.

I want to express my deepest gratitude to my beloved wife Giek Lian for her support and patience throughout this long process. I also warmly thank my parents, sisters, and brothers, who have always supported me in whatever decisions I made. Thanks to my friends in Glasgow and back home in Malaysia for making my life outside research work as enjoyable.

Last but not least, I would like to thank the University of Malaya (Malaysia) for granting me study leave and giving me financial support to pursue my PhD degree.

# CHAPTER 1

## INTRODUCTION

In the 40 years since its launch in the 1960s, the Internet has become a huge platform for numerous applications of different purposes that serve the many and varied needs of different people and organisations. The World Wide Web (WWW or the Web), being the most popular application on the Internet, has shown exponential growth of users in less than 20 years. The increased numbers of Web-based applications and users have had a great impact on performance requirements. The impact can be viewed from the following perspectives:

- Competition between different Web-based application and service providers requires them to out-perform each other and this in turn creates a demand for higher performance.
- Technology advancement makes more complicated functionality possible, which, in turn, increases the processing power required to deliver the functionality.
- Web-based systems have to cope with unpredictable workloads, implying that performance levels of the systems are non-trivial to ensure.
- Having more diversified users means having to meet more diversified user needs, which challenges the developer and systems' capacity.

A problem with referring to performance of computer systems is that there are many metrics for measuring the performance of computer systems, and even the definition of performance itself is open to debate (Lilja, 2000). For example, does the term *performance* used to refer the speed, capacity (size), or accuracy of a computer system? How

performance should be measured or interpreted is also disputable. In this thesis, performance is used to refer to the speed, particularly response time of downloading Web pages from Web sites, namely *Web page response time*, or in shortened format, response time. Response time is the primary focus of this thesis studying Web page performance since it is the metric used to measure end users' experience when using the Web sites. User experience, in terms of response time, is the most straightforward and widely-recognised predictor of user satisfaction (Hoxmeier & DiCesare, 2000). Furthermore, according to Delone and Mclean, as quoted by Kitcharoen (2007), user satisfaction is widely used as a measure of information system success. This is echoed by Palmer (2002) and Bhatti et al. (2000).

There are many factors that affect Web page response time. The network (the Internet) and the Web server are the two main aspects that have attracted much research interest in relation to the study of Web page response time. Examples of studies on the server side include load balancing that deals with distributing workload among a group of servers, as well as content distribution and caching that approach the issue by moving Web content closer to end-users (Iyengar *et al.*, 2002); and automatically tuning server parameters to meet specified requirement levels (Diao *et al.*, 2003). For the network, transmission of data is the major concern. Examples of studies related to the network include the impact of transmission protocols, especially HTTP/1.1, on response time (Cherkasova *et al.*, 2003; Krishnamurthy & Wills, 2000), and transmission methods such as packaging multiple files requested by a users into an object before the object is delivered to the requesting user (Fujinoki *et al.*, 2003). The background literature is discussed in more detail in Chapter 2.

The networking and Web server perspectives on response times view Web servers as entities that receive and transmit bytes of data, i.e. response time is viewed in relation to the network and server activities and conditions. In this thesis, response time is studied from the perspective of Web *pages* in order to provide an alternative view of the metric. Response time is studied as an attribute of the Web page, instead of considering it purely as a result of network and server conditions.

## 1.1 Hypothesis and Thesis Statement

The research focuses on testing the following hypotheses:

1. Response time of downloading a Web page can be decomposed into its constituent components.
2. The particular Web page response time will be affected by identifiable characteristics of the Web pages.
3. It is possible to provide recommendations for resolving or relieving some of the identified performance problems that may cause poor response time based on the identified characteristics that affect Web page response time.

The thesis statement is:

I assert that it is possible to identify the contribution of *generation*, *transmission* and *rendering* to the overall Web page response time and thus identify significant cause(s) of download delay. Furthermore, I assert that it is possible to identify particular characteristics of Web pages themselves that influence this performance metric and to provide advice to Web developers to improve the performance of these individual pages.

A framework that consists of measurement, modelling, and monitoring (3Ms) of Web pages that revolves around response time is adopted to support the research. The *measurement* module enables Web page response time to be measured and is used to support the *modelling* module, which in turn provides references for the *monitoring* module. The monitoring module estimates response time. The three modules are then used in the Web page design and operation phases to maximise the efficiency of Web pages in terms of response time. The framework proposes a systematic way to understand response time through characteristics of Web pages and considers how individual Web page response time can be examined and improved. The framework is presented and discussed in detail in Chapter 3.

## 1.2 Thesis Structure

The rest of this thesis is organised into seven chapters.

- Chapter 2: Background. This chapter covers information and background knowledge related to this research, from the Internet to the Web, and from computer systems performance to response time. Emphasis is put on response time, which is the theme this thesis will revolve around.
- Chapter 3: Problem Statement. This chapter states the problems to be addressed and the framework as well as approaches for addressing the problems.
- Chapter 4: Measuring and Decomposing Web Page Response Time. This chapter introduces two Web page response time measurement methods, which allow response time to be decomposed into three of its constituents, i.e. *generation*, *transmission*, and *rendering*. Examples of measurements are shown and the measurement results are discussed.

- Chapter 5: Modelling Web Page Response Time. This chapter explains the complexity of Web pages from two perspectives: *creation* and *content*. Models that capture the relationship between Web page complexity and response time are discussed. Experiments using one of the methods described in Chapter 4 are carried out to apply and validate the models.
- Chapter 6: Monitoring Web Page Response Time. This chapter exemplifies how Web page *response time* and *responsiveness* can be monitored and estimated based on server access log analysis. A prototype of such a tool is developed to demonstrate how the monitoring could be implemented and the monitoring results could be presented.
- Chapter 7: Making Use of Measurement, Modelling, and Monitoring To Enhance Web Page Performance: An Example. This chapter describes two scenarios demonstrating how the measurement, modelling, and monitoring modules can be used in the Web page design and operation phases to produce Web pages that deliver satisfactory response time. An analyser that examines database queries in Web pages, which may have negative impact on the pages' response times if not used appropriately, is presented to demonstrate how supporting tools can be incorporated into the process to facilitate application of the three modules in designing and maintaining Web pages.
- Chapter 8: Conclusion. This chapter concludes by giving an overview of the thesis. It is then followed by a discussion on the contributions of the thesis. The thesis statement, given in Section 1.1, is reviewed to confirm that assertions of the statement have been fulfilled and the hypotheses supported. The chapter concludes



with a discussion on the limitations of the thesis and a proposal for future work to be carried out based on this research.

## **CHAPTER 2**

### **BACKGROUND**

This chapter provides an overview of aspects pertinent to this research including the background, related work either completed or in progress, and includes other items of interest in the research area. Section 2.1 gives an historical and technical overview to the Internet. Section 2.2 continues with an overview of the World Wide Web (the Web). Section 2.3 focuses on three aspects pertinent to computer system performance, i.e. performance metrics, performance benchmarking, and performance analysis. The section also examines performance in relation to Web-based systems. Section 2.4 explores response time from the perspective of the Web. Section 2.5 discusses techniques for improving response time. Response time measurement methods are grouped into three categories and presented in Section 2.6. Section 2.7 concludes the Chapter.

#### **2.1 The Internet**

##### **2.1.1 Overview and the History**

Before end of the 1960s, electronic computers were expensive and had limited resources. In order to use an electronic computer, one needed physical access to it. In order to share computer resources as well as to communicate using computers, especially among scientists and researchers, the Advanced Research Projects Agency (ARPA) within the government of the United States of America (US) directed and sponsored research on computer networking. The project was named after ARPA and called ARPANET. In 1969, ARPANET linked its first four nodes: the University of California, Los Angeles (UCLA),

the Stanford Research Institute (SRI), the University of Utah, and the University of California, Santa Barbara (UCSB).

ARPANET was based on a packet switching network architecture where data was sent in discrete units called packets across the network. Packets can take different routes and proceed independently on the network and be reconstructed at their destination. Each packet contains information about the address of the destination to which it will be forwarded. Network infrastructure is shared among its users. Packets from different users are transmitted and stored concurrently in network devices.

In this context, packet switching is different from circuit switching. In circuit-switched networks, such as telephone systems, a dedicated communication path needs to be established through the nodes of the network between communicating parties. This sets up a virtual circuit, which is actually a logical channel temporarily reserved over the connection for the communication to take place. Each device along the path will be notified before the communication begins and resources such as buffer space will be reserved to support the circuit. Data is transmitted from the source to the destination along the dedicated path. In packet switching, on the contrary, no network path or resources are reserved in advance of the communication. Packets take ad-hoc paths to reach their destinations. Figure 2.1, adapted from Crovella and Krishnamurthy (2006), shows the difference between circuit switching and packet switching.

The ARPANET grew to about 100 nodes in 1975. In the same year, ARPA handed it over to the Defence Communication Agency of the US for operational management. ARPA also began research on the use of packet switching over other networks such as radio and

satellite communication. Further work from the research had allowed packets to move among wired, satellite, and radio networks by the late 1970s. This idea was realised largely due to the development of Transmission Control Protocol (TCP) and Internet Protocol (IP) and later the acceptance of the protocols as world-wide standards for data communication.

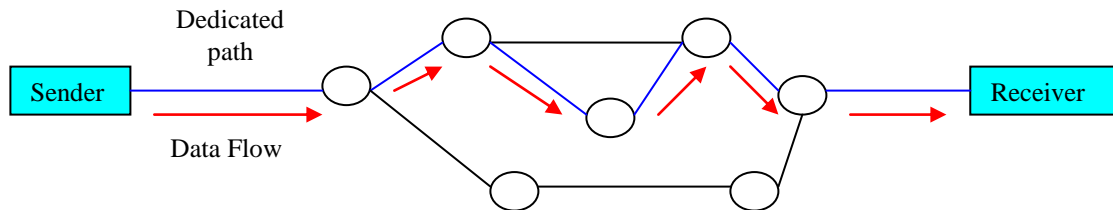


Figure 2.1(a) Data Flows in a Circuit Switched Network

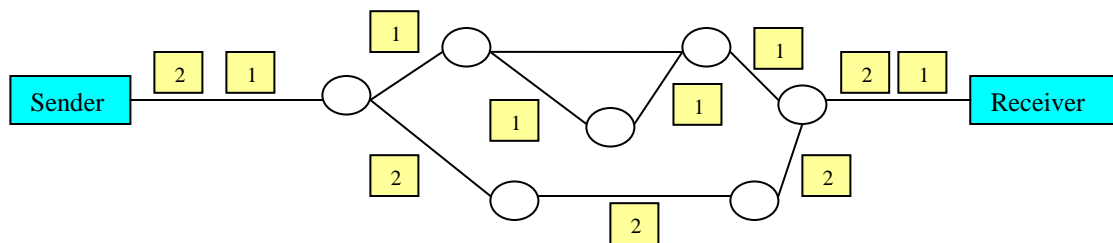


Figure 2.1(b) Data Flows in a Packet Switched Network

TCP/IP uses the client-server model of communication. A client is a computer user that makes requests and the server is another computer in the network that provides the requested services. TCP/IP communication is mainly point-to-point, meaning that each communication proceeds from one point to another point in the network. The sending and receiving points are called end systems. TCP/IP communication is also stateless because each client request is considered independent from previous ones. Such stateless communication allows the same network path to be shared among different users freely. It has to be noted that the TCP layer itself is not stateless as the TCP connection remains open until all packets in a message have been received and the message has been fully reassembled.

TCP and IP form a two-layer program, with TCP as the higher layer. TCP disassembles a message into packets that are transmitted over the network and received by the TCP layer at the other end of the network that reassembles the packets into the original message. IP on the other hand, handles the addressing part of each packet so that it gets to the right destination. TCP and IP will be explained in more detail in Section 2.1.2.

Allowing packets to move across different networks led to the emergence of local area networks (LANs). A LAN is a data communications network that is geographically limited to a single building or a cluster of buildings (Stallings, 2005). The LANs connect devices within a small geographic area and can be linked to the ARPANET through gateways. A gateway is a point in a network that acts as an entrance to another network, and vice-versa. It may contain devices such as protocol translators to perform protocol conversions to allow networks with different protocols to interoperate. A gateway also checks the address of the data packet and forwards it to the destination indicated by the address.

There were two initial motivations behind LANs (Stamper, 1999):

- To overcome computational limitations of a single computer, normally a mainframe, by establishing a networked resource sharing computing environment consists of multiple computers.
- To provide a high speed yet secured departmental computing.

With few less expensive mini-or-microcomputers connected to the mainframe to form a local network, not only the computing power can be improved but also resources such as disk storage and printer can be shared to reduce the cost. LAN gains more advantages over years with the use of group oriented software, the support of communication among workers, better management control, cost effectiveness, and downsizing in which LANs

replaced large computer systems. Figure 2.2 illustrates configuration of LANs in a three tier architecture, adapted from Stallings (2005).

Nodes on the ARPANET and LANs are connected by routers and links. Links physically move packets from one place to another. Routers are special cases of gateways. Routers receive packets from incoming links and place them on outgoing links, in order to forward them to their destinations. Different organisations own and manage routers and links for their LANs. In this sense, each LAN is indeed a separately managed network. These LANs may be operated with different purposes, configured on different network topologies, and running on different technologies. Being connected to the ARPANET, these heterogeneous LANs must be able to communicate with each other using compatible software to allow the connectivity of the Internet as a whole. Compatibility of the software became a crucial problem when the ARPANET grew as more LANs were connected to it.

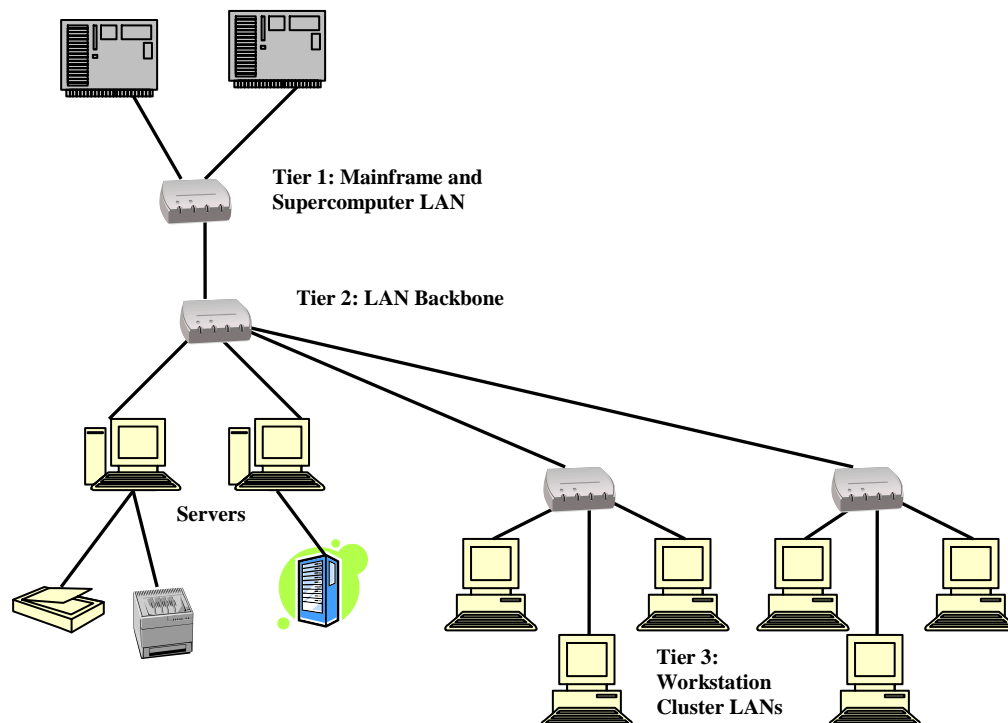


Figure 2.2 Three-tier LAN (Stallings, 2005)

In order to overcome this problem, the network was split into some autonomous systems (ASs). An AS may consist of more than one LAN. Each AS contains a set of links and routers under the same administration. Management of routers within each AS became more independent from others. A set of routers in the ARPANET that formed an AS played a central role called the backbone. A backbone is the central portion of the network that interconnects the other networks and tends to carry traffic over the longest distances. Having the ARPANET backbone connected ASs managed by universities, companies and other institutions exemplifies the ARPANET architecture by the mid-1980s. The idea of viewing the system as an *Internet*, i.e. a network of networks, also started to emerge around this time.

Another problem that needed to be resolved when the Internet was growing was addressing. Each source and destination on the Internet was assigned a 32-bit IP address (before IPv6, see Section 2.1.2 for more details), usually written as four decimal numbers, which was interpreted as two components:

- A network number that identifies a particular network on the Internet.
- A host address that identifies a specific host on that network.

To send a data packet from its source to the destination, routers along the path had to keep a master table that contained addresses of all the networks on the Internet. When the number of network addresses on the Internet increased, the table at each router had to be updated accordingly so that the contents of all the tables were up-to-date and consistent. Indeed, the number of network addresses increased exponentially. As a result, the management of the table was cumbersome. Moreover, as the number of hosts on the Internet grew, it became more difficult for human beings to remember or associate the hosts with their numeric addresses.

The concept of a Domain Name System (DNS) was thus introduced to address the problems. A domain is a group of hosts under the administrative control of a single entity. Each domain is given a name rather than referred by a numerical number (IP address). For example, *edu* refers to the domain of educational institutions while *com* is the domain of commercial organisations. Domains are hierarchically organised as shown in Figure 2.3. DNS is used in such a way that each higher level domain name is prefixed with a subordinate domain name to form the address of a host on the Internet. For instance:

- uk is the country code top level domain (TLD) for the UK.
- ac.uk is the domain for academic institutions in the UK.
- gla.ac.uk is the domain for the University of Glasgow in the UK.
- dcs.gla.ac.uk is the domain for the Department of Computing Science at the University of Glasgow.

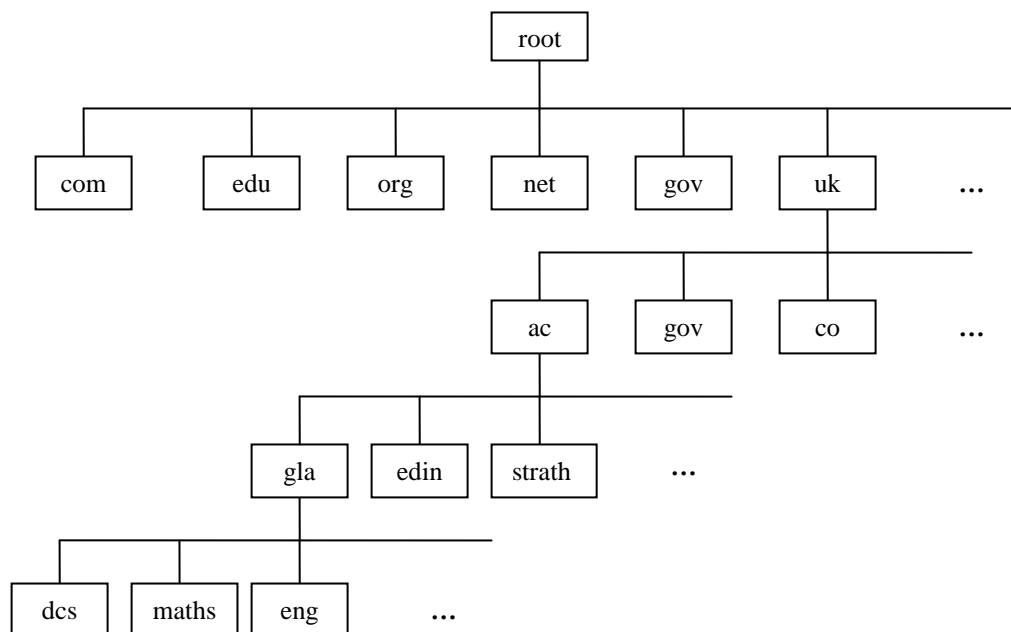


Figure 2.3 Hierarchical Structure of Domain Name System



Domain names and relevant information are stored in a DNS database, which, in turn, is held by a name server. Resolvers are the programs used to extract information from name servers, typically to resolve a given domain name requested by the client into the corresponding IP address.

In January 1983, TCP/IP became the only approved protocol on the ARPANET, replacing the earlier Network Control Program (NCP) protocol. In the same year, the US Department of Defence also split off the military network MILNET from the ARPANET. The ARPANET was becoming a non-classified research network. In 1984, The US' National Science Foundation (NSF) constructed a university network backbone that later became the National Science Foundation Network (NSFNET). This was the first TCP/IP wide area network (WAN) that was put in operation and it marks the birth of the Internet. A WAN is a computer network that covers a broad geographical area, as opposed to a LAN that is setup within a limited geographical area, like a building. The NSFNET became the network backbone when the ARPANET was retired in 1990.

There were other networks connected to the NSFNET. These networks included research-based and educational networks as well as networks outside the US. The networks provided different services, such as bulletin board system (BBS), emails, and file transfers. One of such systems was the Unix to Unix Copy (UUCP) developed in late 1970s. Some of the networks operated on protocols other than TCP/IP, for example International Telecommunication Union's Telecommunication Standardization Sector (ITU-T)'s X.25 protocol suite. X.25 defines the standards for WANs to use the telephone or Integrated Services Digital Network (ISDN) system as the networking hardware, where ISDN is a circuit-switched telephone network system. The ability of TCP/IP to work over these pre-

existing communication networks allowed them to be connected to the NSFNET easily using interfacing gateways and resulted in the growth of the NSFNET.

Until the mid-1980s, the ARPANET and networks that were based around it were government funded and restricted to non-commercial uses such as research, military, and education. Commercial use was forbidden. By the late-1980s, more companies were participating in research projects related to the ARPANET or providing services to those involved in the projects. These companies sought to connect their networks in order to use services such as electronic email and file transfer. In 1987, the first commercial Internet Service Provider (ISP) was set up and soon followed by many others. An ISP provides its customers access to the Internet and related services for a monthly or per-use charge. Thus, the transition of the network to commercial use had begun.

By early 1990s, NSF had changed its policy from prohibiting commercial institutions from using networks paid for by NFS to encouraging the development of intermediate regional networks that connected many enterprise networks to the NSFNET backbone. As a result, a three-tiered network organisation appeared, i.e. the backbone, regional networks, and enterprise networks. NSFNET itself was turned into a non-profit corporation to operate and improve the backbone. However, since many commercial ISPs existed by 1995, the NSFNET backbone was no longer needed and thus withdrew and retired. Since then, the Internet has had no single default backbone but different ISPs run their own backbone networks to compete with each other with the agreement to carry each others' traffic.

Outside the US, connection to the ARPANET was rare at first. European developers were more concerned with developing the X.25 networks. Similar to TCP/IP, X.25 is also

referred to as a packet switching protocol. However, X.25 is a connection oriented protocol as compared to IP which is a connectionless protocol. Address of the destination is stored within each IP data packet and that information is used for routing at network switching nodes on an ad-hoc basis. X.25 on the other hand, uses packets that carry routing information to first establish a connection between the communicating parties. Once the connection is established, addressing will be taken care of by network switching notes. That means addressing information is not stored in data packets flowing between the communicating parties.

The first ARPANET connection outside the US was established to Norwegian Seismic Array (NORSAR) in Norway in 1973. It was only in the mid-1980s that TCP/IP was more widely used in Europe. The European Organization for Nuclear Research (CERN) established a TCP/IP network that was isolated from the rest of the Internet. Later, the Internet expanded in Europe across the existing UUCP networks which allowed remote execution of commands and transfer of files, emails and BBS between computers. In Australia, the Australian Academic and Research Network (AARNet) was formed in 1989 by the Australian Vice-Chancellors' Committee to provide a dedicated IP based network for Australia and a connection to the ARPANET. In Asia, the Internet also began to enter the arena by end of 1980s and the early 1990s. Nowadays, the Internet has evolved into an exponentially growing world-wide interconnected collection of local and wide area networks.

The emergence of the World Wide Web (WWW or simply the Web) is a major force that made the Internet known to the general public. The Web made the Internet easier to use and the invention of the graphical Web browser enticed users from areas other than the

traditional academic, government, and research users. More details about the Web will be given in Section 2.2.

### **2.1.2 Packet Switching and Internet Protocol Suites**

The Internet is a packet switched network. Packet switching is the dominant communication paradigm in the fields of computer networking and telecommunications at the present time. As mentioned before, in packet switched networks, packets are transmitted between nodes such as bridges, routers, repeaters, or switches over data links shared with other traffic. Routers, repeater, bridges, and switches are devices to support data transmission across the network:

- A bridge is a device that connects a LAN to another LAN that uses the same network protocol. It uses broadcasting technique to send messages out to every address on the network and accepted only at the intended destination.
- A router is a device that connects two or more networks and determines the best path to forward data packets toward their destinations, through a process called routing. While a bridge connects only two LANs with the same protocols, a router can connect a variety of LANs and WANs with different protocols. Since a router involves in the translation of protocols among dissimilar networks, it is therefore a kind of gateway.
- A repeater is a device that receives a weak or low-level digital signal and regenerates the signal at a higher level so that the signal can be transmitted over a longer distance without degradation. A hub is a form of repeater with multiple ports that are connected to other devices in the network.

- A switch is a device that directs incoming data from any of multiple inputs to the specific output that will take the data toward its intended destination. This is different from a hub which receives an input and broadcasts it to all the output lines attached to it.

A packet is a formatted information block carried by communication networks. Using packets allows larger messages to be chunked into smaller pieces and transmitted more reliably and efficiently over networks. A packet consists of three elements: a header, a payload, and a trailer. The header indicates the beginning of the packet; the payload is the information carried by the packet; the trailer marks the end of the packet. The header contains information to relay a packet across the network to its intended destination as well as information for the packet to be parsed, error-detected, and reassembled in the correct order. Both TCP and IP define the format for their headers. The latest standard for IP is IPv6 released in 1996 by the Internet Engineering Task Force (IETF), which includes 128-bit source (sender) and destination (receiver) address fields, as compared to 32 bits used in the earlier IPv4 standard. Using 128 bits for the source and destination fields supports the need for more addresses on the Internet.

The operation of packet-switched networks in general or the Internet in particular, relies on different communication protocols organised into a protocol stack, which is called the Internet protocol suite or the TCP/IP protocol suite. The Internet protocol suite consists of four layers. Each layer specifies protocols pertinent to certain problems about the transmission of data, such as format of data packet and physical interface between data transmission devices, provides services to the upper layer protocols, and uses services from protocols at the lower level to help to accomplish their aims. Upper layers are logically

closer to the user and deal with more abstract data. Lower layers, on the other hand, translate data into forms that can be physically transmitted. The four layers are:

- Layer 1: Network Access Layer - Describes physical equipment required for communications and the signalling used on that equipment, as well as the low level protocols using that signalling. Examples include Fiber Distributed Data Interface (FDDI) and Wi-Fi.
- Layer 2: Internet Layer - Defines IP addresses and routing schemes for transmitting packets from one IP address to another. IP operates at this layer.
- Layer 3: Transport Layer - Defines protocols for flow control and establishing connection, such as TCP.
- Layer 4: Application Layer – Defines higher level protocols to deal with data abstractly, such as Hypertext Transfer Protocol (HTTP).

This original four-layer Internet protocol suite has evolved into a five-layer model that splits Layer 1 into a Physical layer and a Network Access layer, corresponding to the physical layer and data link layer of the seven-layer Open Systems Interconnection (OSI) Basic Reference Model, or OSI Model for short. The seven layers of the OSI Model are physical, data link, network, transport, session, presentation, and application. Figure 2.4 compares TCP/IP protocol suite with the OSI model (Stallings, 2005).

Relevant protocols at each layer will append control information on top of the data to be sent and control information appended by protocols at upper layers. The control information is required to ensure that the data is sent correctly to its destination. The control information will be interpreted by the corresponding protocol at the same layer at the

destination. The control information combined with the data block passed down to the next lower layer is known as a protocol data unit (PDU). Figure 2.5 shows PDUs in the TCP/IP architecture.

OSI	TCP/IP
Application	Application
Presentation	
Session	
Transport	Transport (host-to-host)
Network	Internet
Data Link	Network Access
Physical	Physical

Figure 2.4 Comparison between TCP/IP Protocol Suite and the OSI Model (Stallings, 2005)

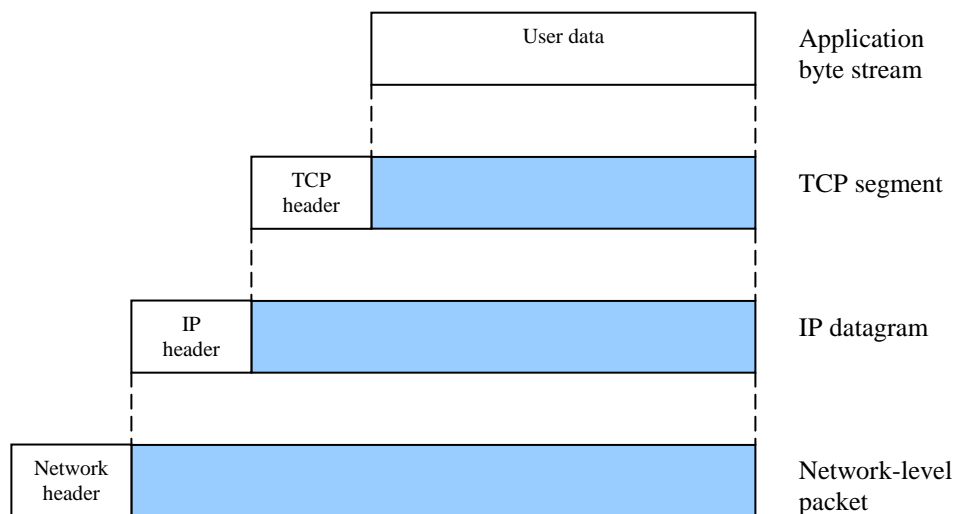


Figure 2.5 Protocol Data Units (PDUs) in the TCP/IP Architecture (Stallings, 2005)

In order to allow a communication to take place over the Internet, a computer or communication device, called the host or end system, is connected to some sort of network using a network access protocol, such as the Ethernet logic. This protocol enables data to be sent from the one host to another across the network, or to a router if the receiving host is on another network. Every entity in this overall system must have a unique address to identify it and allow data to be delivered to the proper host. This address is used by IP for routing and delivery. Therefore, IP is implemented in all end systems and intermediary routers. Apart from this unique global IP address, each application within a host must have a unique address within the host itself. This address, referred to as the port, facilitates a host-to-host protocol, i.e. TCP, to deliver data to the proper process or application on the right host. TCP is implemented in the end systems only. In this sense, TCP and IP form a two-level addressing system that ensures the reliability of data delivery from one host to another. It is important to note that TCP does not need to be told about the destination's IP address and IP will not be notified of the destination's port number. This is in accordance to the Internet protocol suite's layered architecture which supports scalability of the Internet. Figure 2.6 illustrates a communication between two hosts using TCP/IP.



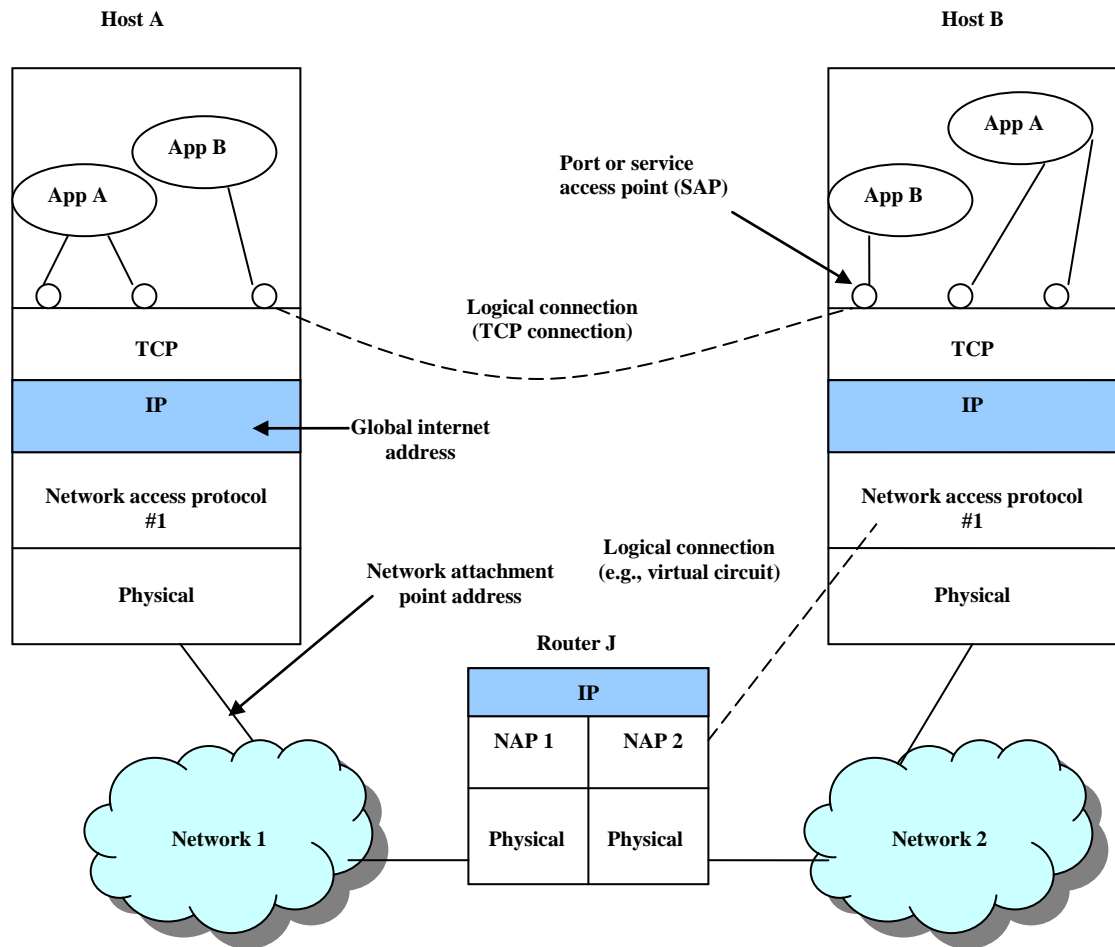


Figure 2.6 TCP/IP Concepts (Stallings, 2005)

## 2.2 The World Wide Web

The Internet provides a space for many services such as electronic mail, file transfer, and the World Wide Web (WWW or simply the Web). The Web is often ambiguously used as a synonym for the Internet since it is the most common and significant service on the Internet. Like other services built on top of the Internet, the Web relies on the TCP/IP Protocol for communication among connected computers. Apart from that, the Web also has its own layer of Hypertext Transfer Protocol (HTTP) for retrieving documents, Hypertext Markup Language (HTML) for laying out pages of information and Uniform Resource Identifier

(URI) as its global naming scheme. These are the three basic technical components that comprise the Web.

The Web originated from the European Laboratory for Particle Physics operated by CERN where a system for exchanging information globally among researchers at the institute was needed (Wilde, 1999). Tim Berners-Lee was working at the institute at the time and he produced a paper called Mesh in 1989, which outlined such a system (Berners-Lee, 1989). The paper formed the ground for a prototype and the project was officially named World Wide Web by the end of 1990. The success of the Web was ensured by the release of Mosaic X, a usable and powerful graphical browser developed by Marc Andreessen for the National Center for Supercomputing Applications (NCSA) in 1993. Through publications such as New York Times and the Economist, the Web was introduced to the general public and started to gain increasing popularity.

In 1994, NCSA and CERN founded the World Wide Web Consortium (W3C) as the international association to regulate the evolution of the Web. Later, CERN decided not to continue to play its role and the responsibility was taken over by the French National Institute for Research in Computer Science and Control (INRIA). W3C develops and maintains standards such as HTTP, HTML, and style sheets to ensure effective communication and storage of information on the Web.

The popularity of the Web also led to the so-called dot-com bubble covering roughly 1995-2001. The Web brought the possibilities of free publishing, instant worldwide information dissemination and receiving, two-way or even group communication, and bringing together unrelated buyers and sellers, or advertisers and clients, seamlessly and with very low cost.

Such possibilities promised a new Web-based business model, called e-commerce. Many Internet-based companies, commonly referred to as dot-coms, were established based on this new business model and the capital for establishing these companies often came through public offerings on the stock exchanges. A combination of rapidly increasing stock prices, individual speculation in stocks, and widely available venture capital because of low interest rates, resulted in an over exuberant environment where many of the companies focused on increasing market share rather than the business itself. Eventually, the dot-com bubble burst in the Spring of 2000 and through 2001, giving way to a more measured and grounded business model today.

In recent years, the term Web 2.0 has been used to describe a perceived second generation of Web based services. Even though the concept is relatively new and its actual meaning is still open to debate, O'Reilly (2005) summarised some characteristics of Web 2.0 which may best describe what Web 2.0 is.

- The Web as the platform to serve applications to end users.
- Harnessing collective intelligence where direct involvement of Web users contributes to the success of the Web based applications or services. For example, Wikipedia (<http://www.wikipedia.org>) is an online encyclopaedia that allows Web users to add and edit its entries.
- Data is the core of Web based applications and services and database management is a core competency of Web 2.0 companies.
- Software is delivered as a service rather than a product. Users are treated as co-developers of such services. New features of software are released to the market on a

monthly, weekly, or even daily basis, creating 'perpetual beta' phenomena. Thus, it denotes the end of software release cycle.

- Lightweight programming models that promote loosely coupled and syndicated systems formed by assembling features from different developers. This also supports higher level of software or code reuse.
- Software runs on multiple devices instead of limited to the PC platform.
- Rich user experiences introduced by many new Web applications and reimplementations of PC applications in the Web form.

Conceptually, the Web is a collection of various information resources, including text documents, images, and multimedia. Uniform Resource Locators (URLs) are used to locate these resources on the Web. URL is a special form of URI that consists of a scheme and a scheme-specific part. The scheme-specific part contains information such as user and password (normally omitted), host (domain name or IP address), port number and URL path. The scheme determines the interpretation of different components of the scheme-specific part. Examples of the scheme include http, ftp, https, and telnet.

On the Web, servers are used to store information resources. A client program called a user agent retrieves information resources from the server using their URLs. The user agent is normally a Web browser that renders the information on a computer display. Information is presented in the form of a Web page and a collection of related Web pages forms a Web site. Web pages can be linked to each other by means of hyperlinks, or by means of anchors within a Web page itself.

Before a server can serve requests from the clients, it must be bound to a port (port 80 by default) for setting up connections with the clients. To request a Web page, the client first has to locate the hosting server by doing a DNS lookup, which will resolve the server's name into the corresponding IP address. The next step is to establish a TCP connection with the server using a three-way handshake. The three-way handshake occurs as follows:

- (1) The client sends a SYN packet to the server.
- (2) The server replies with a SYN-ACK packet.
- (3) The client returns with an ACK packet.

Each SYN and ACK packet holds a 32-bit sequence number and the numbers are used in later stage to identify the order of the data bytes sent to the client.

After the connection is set up, the client will send the request for one Web page, denoted by its distinct URL. The server will receive the request, find the requested page, and respond to the requesting client by delivering the page as segments of data bytes. These segments will include information necessary for transmitting them to the requesting client as well as rearranging them into correct order at the client side. While the responses are being received, the client Web browser will parse through and render them so that the requested page can be displayed correctly on the browser. If there are embedded objects in the page, such as images, style sheets, audio and video clips, separate requests will be sent to obtain these objects. Figure 2.7 shows the sequence of communications on the Web between a client and a server using the HTTP protocol to download a Web page. DNS lookup is not shown in the figure as it is normally handled by another server.

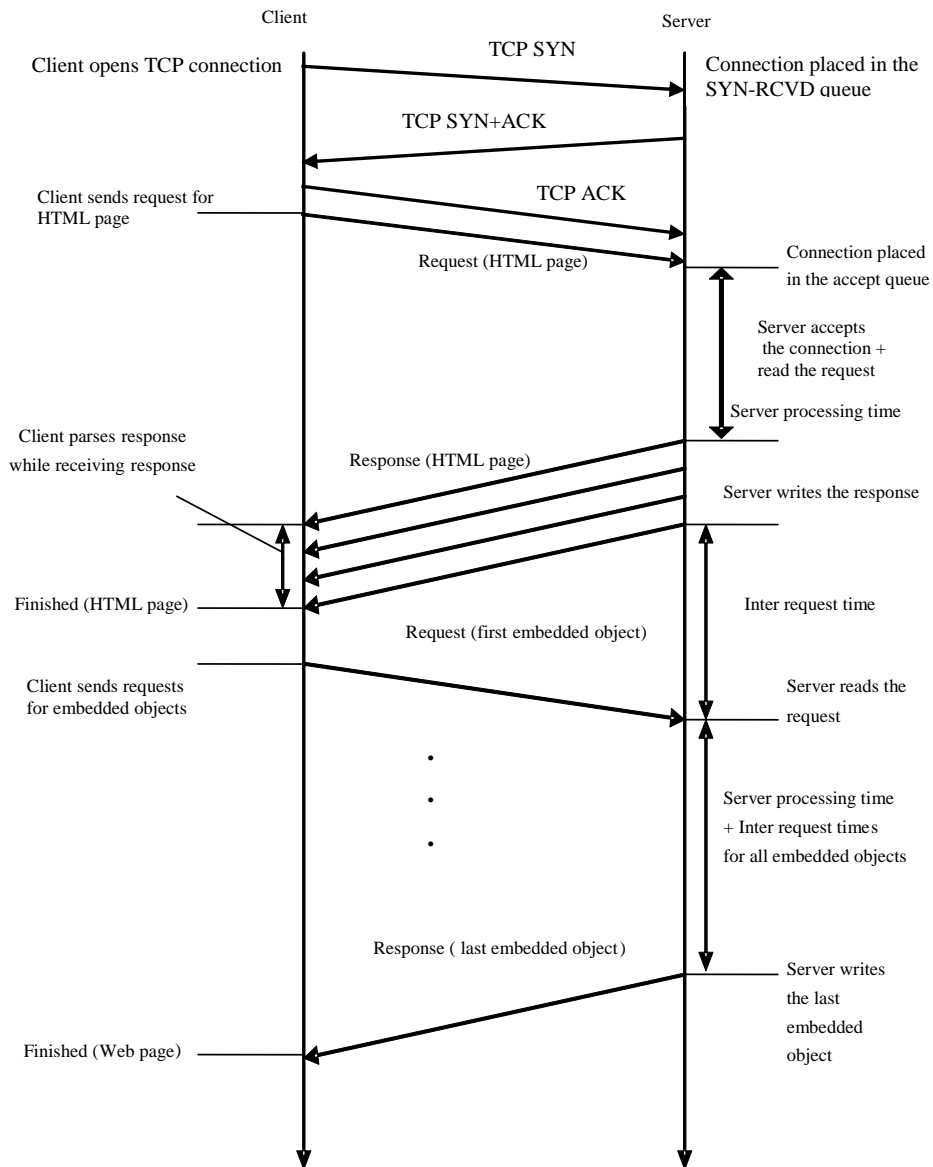


Figure 2.7 HTTP Request-response Sequences for Downloading a Page

Nowadays, Web sites are not only spaces where information is shared, but also facilitate the provision of different kinds of services. Thus, a Web site is indeed a system that utilises the Internet as its underlying infrastructure and the Web as its platform. Quality of service (QoS) issues related to normal computer applications apply to Web-based systems too. These issues include, but not limited to, reliability, usability, efficiency, security, and consistency. Since the Web involves geographically distributed and heterogeneous systems

and users, these quality issues are more critical for Web-based systems. Meanwhile, some quality measures become more significant on Web-based systems than for systems running on a single computer. One of the quality measures, which is the focus of this research, is the systems' performance in terms of response time.

## **2.3 Performance**

### **2.3.1 Performance Metrics**

Computer systems' performance can be described by different metrics and analysed using different techniques. There are many metrics for measuring performance of computer systems, and the definition of performance itself is debatable. Equating performance with speed at which tasks are completed by a system is a common yet misleading concept. In fact, performance of a system can also refer to how efficiently the system uses resources such as the computer's central processing unit (CPU), memory, or disk storage. Other views of performance include start-up time, i.e. how quickly a system launches, and scalability, i.e. how does the system's performance change under varying loads. Ironically, performance may also involve a subjective measure that reflects the user's perception of the system, particularly in terms of the speed or responsiveness. It should be noted, however, that user perceptions are outwith the focus of this research.

Adopting different definitions of performance will yield different metrics to be designed and used for performance analysis. Among the performance metrics are:

- (a) Response time or latency: The time between the user finishes a request and the system starts or completes the execution. Further details are discussed in Sections 2.4 to 2.6.

- (b) Throughput: In general, it refers to the number of jobs a computer system completes per unit of time. In the context of data communication, it refers to the amount of data, normally measured in number of bytes, flow through the network in a given time.
- (c) Utilisation: The fraction of time a computer resource is busy servicing requests over its available time. A higher utilisation level means a higher load imposed on the resource and a higher tendency of performance declining, while a low utilisation level may imply a waste of the resource as it is under utilised.
- (d) Availability: Downtime or uptime of a system, measured as mean time between failure (MTBF) or mean time to repair (MTTR). MTBF denotes the duration in which the system is functioning between a system breakdown and the following one. MTTR is the average time that a system will take to recover from a failure. Another measure, maintenance-free operating period (MFOP), is the period of time during which there is not need for system maintenance. Both MTTR and MFOP take into account of repairs or maintenance required to diagnose or rectify problems that do not necessarily cause a system fails to function completely.

As a guideline, Jain (1991) summarised performance requirements into five criteria, i.e. *specific, measurable, acceptable, realisable, and thorough*, or SMART as the acronym (p. 42):

- Specificity precludes the use of words like ‘low probability’ or ‘rare’.
- Measurability requires verification that a given system meets the requirements.
- Acceptability and realisability demand new configuration limits or architectural decisions so that the requirements are high enough to be achievable.



- Thoroughness includes all possible outcomes and failure modes.

It is obvious that the four metrics listed above fulfil the five performance requirements criteria and provide appropriate indications to different performance aspects of a system.

### 2.3.2 Performance Benchmarking

Performance metrics are keys to performance benchmarking. Performance benchmarking is a measuring process used to compare systems (Kotsis, 2006). Systems under test (SUT) are tested under a standardised workload defined by the benchmarks. Performance of the SUTs is characterised by the metrics selected. For Web-based systems, throughput (X) and response time (R) are the two most often used metrics. In general, the desired outcome is to achieve maximum system throughput while meeting response time requirements. Figure 2.8 shows throughput and response time as functions of workload (Kotsis, 2006). As workload increases, both throughput and response time will increase as well. In benchmarking, a service level agreement specifies nominal system capacity and/or maximum acceptable response time.

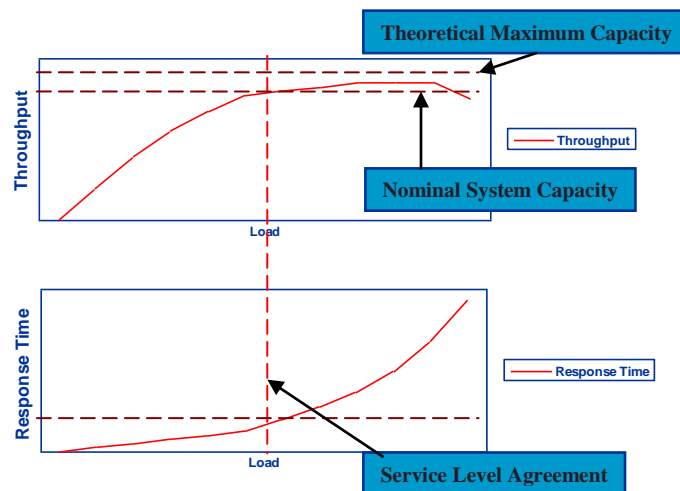


Figure 2.8 Throughput and Response Time as Functions of Workload (Kotsis, 2006)

Examples of benchmarking for Web-based systems include TPC-W and SPECweb99. TPC-W by the Transaction Processing Performance Council measures performance of different servers carrying out different functions, such as Web server, image server, database server and Web cache (Smith, 2001). It focuses on the number of Web interactions processed by the servers per second. Standard Performance Evaluation Corporation's SPECweb99 on the other hand, is a software benchmark used to evaluate Web server performance. It measures a system's ability to act as a Web server for static and dynamic pages (SPECweb99, 2000). The focus of SPECweb99 is the maximum number of simultaneous connections requesting predefined workload that a Web server is able to support while still meeting specific throughput and error rate requirements.

### **2.3.3 Performance Analysis**

#### **2.3.3.1 Categories and Issues**

Performance analysis techniques are very much easier to categorise than performance metrics. There are three fundamental performance analysis techniques: *analytical modelling*, *simulation*, and *measurements of existing systems*.

- Analytical modelling requires the construction of a mathematical model as an abstract representation of the real system. It provides a quick insight into the system's performance by focusing on only a few salient system details. The analysis results tend to be less accurate than the results produced from the other two categories of techniques, but the technique provides a good platform for the validation of results from simulation and measurement.
- Simulation also relies on the construction of a system model, but the model is a specialised computer program that captures only features of interest to the

simulation study. Such simplification of system behaviour eases the design, development and modification of the model but limits the accuracy of its results.

- Measurement is straightforward – monitors or benchmarks existing system of interest without simplifying assumptions. Thus, this produces more accurate results than the other two techniques, but is less flexible because it is difficult to change the measurement parameters. Table 2.1 compares the techniques (FrontRunner, 2002; Lilja, 2000).

Table 2.1 Comparison among Performance Analysis Techniques

Characteristic	Performance Analysis Technique		
	Analytical	Simulation	Measurement
Flexibility	High	Medium	Low
Cost	Low	Medium	High
Accuracy	Low	Medium	High
Time Needed	Small	Medium	Large
Skill Level	High	Medium	Medium
Scalability	Low	Medium	High
Tools	None or calculation software	Simulation software and/or programming language	Load generators and instrumentation
Can be used	At any time	At any time	When working system is available
Metrics provided	Limited	Any	Varies with instrumentation
Quantifies variability	No	Yes, with multiple runs	Yes, with multiple runs

As claimed by Kotsis (2006), performance analysis (and management) should become an integral part of the development and operation of Web applications. There are three issues related to performance analysis and management. The “classic” performance analysis deals with performance metrics and requirements. Capacity planning tackles the problems regarding the service level requirements of the application. Stress tests examine maximum load that an application can handle.

### **2.3.3.2 Goals and Steps**

In experimental computer science, performance analysis can be thought of in terms of measurement, interpretation, and communication of a computer system's speed or capacity.

Performance analysis has six major goals (Lilja, 2000):

- (a) Compare alternatives to find the quantitatively best performance solutions.
- (b) Determine the impact of various system features on performance.
- (c) Tune the system to achieve the best performance.
- (d) Identify performance compared to the history of the system, customer's expectation, or the competitor's systems.
- (e) Performance debugging that views performance loss of a system as a result of design fault and attempts to detect and remove it.
- (f) Set expectations for the next generation systems.

In general, five steps will be involved in performance analysis (Kotsis, 2006). They are:

- (a) Define the system to be analysed and the metrics for testing.
- (b) Characterise the workload for testing.
- (c) Select techniques and tools for analysis and run the experiments.
- (d) Represent and interpret analysis results.
- (e) Improve and optimise performance based on the analysis results.

### **2.3.4 Performance and Web-based Systems**

During the early years of computing, performance was more concerned with computer hardware, such as the CPU. Relevant performance metrics in this case are clock rate, millions of instructions executed per second (MIPS) and execution time. Most of these metrics are pertinent to the efficiency of the hardware at execution. As computer hardware

is developing and enhancing at a rapid rate, it is able to support more diversified types of software and less likely to become the bottleneck for a system's performance. On the contrary, software is evolving at a much slower rate than hardware. Nevertheless, the complexity of software is also increasing. Software performance is therefore becoming more critical than hardware performance as compared to those early years.

In the context of Web-based systems, both hardware and software could influence the systems' performance to a large extent. User perception is another factor to be considered when dealing with performance issues. Performance issues are thus more complicated in this environment. Meanwhile, the Web has become one of the most significant computer applications, on one hand, and the number of Internet users is growing exponentially on the other hand. Both factors have a great impact on performance requirements of Web-based systems. As a result, performance issues for Web-based systems have been attracting much research interest in recent years (Killelea, 2002). Sections 2.4 to 2.6 discuss one of the issues, i.e. response time in greater detail.

Performance issues are also a major concern of the Web industries and Web practitioners. Attracting more visitors to a Web site, encouraging them to stay longer and use the services or buy the products sold are the ultimate goals of Web site owners. These goals will hardly be realised if the system exhibits poor performance. One of the important performance issues for e-commerce Web sites is response time. It is a contributing factor to customer loyalty in B2C e-commerce and unreasonably long response times tend to repulse customers from a Web site (Debaraj *et al.*, 2003). As quoted by Chow (2001), a study conducted by the Boston Consulting Group and reported in ZDNet's PC Magazine showed that 48% out of 12 000 online customers surveyed gave up trying to buy some products

because the Web pages took too long to load. Chow's study also found that sequences of delays such as from short to long delay or vice versa, affect the users' level of annoyance.

As mentioned before, there are different metrics to describe different aspects of system performance. Choosing a particular metric to be used depends on the nature of the system and the goals of performance analysis. Since the success of Web-based systems depends mostly upon user satisfaction, the best metric to study the performance of a Web-based system should be one that describes system performance from the users' perspective. The one metric that impacts all users is response time (Broadwell, 2004).

## **2.4 Response Time**

Response time, or latency, is the time elapsed between two related events – a start and a stop event. In Web-based systems, there are two different definitions of response time from the user's perspective, distinguished by the descriptions of the stop event. The definitions are:

- (a) The time elapsed from the moment the user requests a Web page until the requested page is displayed *in its entirety* on the user's machine. This is the definition adopted for response time in this research.
- (b) The time elapsed between the start of the request and beginning of the response, i.e. the page, starts displaying on the user's machine. In this research, this is referred to as *responsiveness*, and is discussed more thoroughly in Section 6.3.

The first definition proposes a more technical view of completion of a transaction between the client and the server. The second definition takes the user's psychological perception of

the server response into consideration. Based on the second definition, it is assumed that the user will not only feel more comfortable when seeing the partial display, but can also start to peruse the browser display while waiting for the rest of the page to display. This is in accordance with the concept of response time as a user-perceived attribute. However, the correctness of such assumptions relies on the content of the Web page and user needs. It might be the embedded and more slowly loading objects that are significant to the user rather than the pure text being displayed first. Moreover, there is no guarantee of completion of the response even though partial responses have been received and are being displayed at the client browser.

Regardless of which definition is adopted, times for DNS lookup, TCP connection set up, requesting and retrieving the Web page and its embedded objects will constitute user-perceived response time. This is a view of response time components from the chronological order of the activities that make up a transaction. Spatial-wise, response time can be decomposed into three constituents, namely:

- Network transmission,
- Server-side request processing and response generation, and
- Client-side request sending and response displaying.

Decomposing response time into its constituents helps us to identify and isolate possible bottlenecks of Web performance. For example, the ratio of server side request processing and response generation time to the overall response time gives a good indication of the underlying performance of the server. A further analysis may decompose server side request processing and response generation time into components such as time taken by

each request to wait in the queue, processes to generate dynamic content, and database queries. This provides a good foundation for rectifying problems that delay responses and suggesting appropriate improvement actions to be taken.

One of the issues about response time that draws the attention of Web suppliers, especially e-Commerce practitioners, is how long the users are willing to wait for a Web page to be downloaded before giving up. Nielsen (1999) put a very high regard on the importance of response time or download speeds as “*the single-most important design criterion on the Web*” (p. 67). However, there are many different views of how long the maximum acceptable response time should be. For instance, a decade ago, Nielsen (1997) had proposed that 10 seconds as the maximum response time for downloading a Web page. Zona (1999) proposed a lower threshold of 8 seconds.

The arguments are best described through the summary made by Nah (2003). According to Nah’s summary based on others’ studies, even though long download or response time of Web pages has been a consistent problem encountered by Web users, it is still controversial as to what constitutes an acceptable waiting time for a typical Web page download. There are different tolerable waiting times (TWTs) for Web page download proposed by different researches, ranging from 2, 4, 8, 10, to 12 seconds. Some even argued that “*there is no difference in users’ frustration levels between 1 and 20 seconds delay, but a difference (with 1 second delay) was observed at 30 seconds delay*” (p. 3). However, “*the average American users that use dial-up connections wait about 30 seconds the first time they look at a new Web page*” (p. 4). This shows a paradoxical situation in which expected response time is measured and studied, as well as what the actual response time is. Nevertheless, the summary was made in 2003 and some of the data given in the summary were obtained



earlier than that. With the percentage of broadband Internet users increasing across the world, emergence of more diversified Web applications and services, introduction of newer Web technologies, and changing user expectations, the scenario is likely to have changed since 2003.

It has to be noted that user perceived Web latency is not only measured by the actual response time but also affected by psychological factors of the user, such as tiredness. The Web-surfing context such as the type of Web-based system or application, the user's Internet connection speed, and types of tasks carried out also affects the user's tolerable waiting time (Nah, 2003). For example, comparing simple information retrieval tasks and online purchasing, it has been found that users are more willing to wait longer for the latter as they have a more vested interest in it (Kotsis, 2006). It is also commonly believed that giving hints or information about waiting time in advance to the users, or retrieval information and status given while the users are waiting for Web pages to download, will increase the users' willingness to wait (Hitz *et al.*, 2006).

Other factors that affect users' tolerable waiting time include the amount of multimedia or graphics available on the Web site/page, incentives or rewards for completion of the task, users' experience, age, gender, personality and culture, availability of alternative Web pages, time pressure, and environmental factors (Nah, 2003).

## **2.5 Improving Response Time**

In spite of many factors that affect user perceived Web latency, response time has the closest and the least arguable relationship with users' perception of the delay. Improving

Web response time is the most straightforward way of improving users' perceptions of a particular Web site's latency. In order to improve response time, different approaches can be adopted. Some of the approaches are discussed below.

### **2.5.1 Caching**

Caching is the most important and most widely used performance improvement technique for Web-based systems (Killelea, 2002). The idea of caching is to keep frequently accessed data at locations close to the clients such as the client browsers or Web proxy servers<sup>1</sup>. Retrieving data from these caching locations will not only reduce transmission time across the Internet, but also reduce workloads imposed on the Web server. Thus, caching trades storage space and currency of content for access speed.

The main issue with caching is to maintain consistency of the cache across the Web. It is a less serious problem with static Web content but for dynamic Web content, caching may deliver outdated information. There are four categories of degrees of consistency between the content at the Web server and those cached, i.e. strong, delta, weak and mutual consistencies (Iyengar *et al.*, 2002). The four categories are:

- (a) Strong consistency. The latest content is always returned by the cache. This requires the client to poll the server to validate the currency of cached objects should they be requested, or the server invalidates or updates all the outdated cached objects even though they are not requested. However, exchanging of messages between the client and the server requires time within which the object may have changed again after the server validates its currency. Message delay on the Internet is also unbounded,

---

<sup>1</sup> A proxy server is an intermediary between end users and the Internet for the reasons of security, administrative control and caching. It is one of the mechanisms to protect an enterprise network from outside intrusion.

making strong consistency hard, if not impossible, to guarantee. Therefore no cache consistency mechanism can be ideally and truly strong.

- (b) Delta consistency. Data returned by the cache is never outdated by more than  $\delta$  time units, where  $\delta$  is a configurable parameter. The value of  $\delta$  should be larger than the network delay between the cache and the server. If the value of  $\delta$  is the same as the network delay or slightly greater than it, delta consistency can therefore be regarded as the practical implementation to achieve strong consistency.
- (c) Weak consistency. A read at the cache may retrieve some previously correct content.
- (d) Mutual consistency. A group of objects are mutually consistent with respect to each other.

The attempt to maintain content consistency can be initiated by the server, the client, or based on an explicit mechanism. Server-driven mechanisms include the server updating the cached content when the corresponding server content is changed. Client-driven mechanisms require the client to poll the server for the latest content at each request, or only for the requests for cached objects that have exceeded a specified time duration. Examples of explicit mechanisms include the use of a group of Web caches that work together to form a cooperative cache, where requested objects not found locally will be retrieved from neighbouring caches before the requests are forwarded to the server. Other examples include the server issuing a lease for each requested object, or group of objects, or even a group of proxies. The lease specifies the duration within which the server will notify the cache should the content change. For the case of updating the cache, the server could use the delta encoding technique where only the changes to the object are sent, rather than the entire object. This will reduce the size of the data transferred and response time as well. A server can also adopt other mechanisms for maintaining cache consistency. For

instance, based on the frequency of the objects being requested, the server can selectively update frequently requested objects (requires more data to be delivered to the caches to overwrite existing objects) but invalidate less popular objects (only notifies the caches that the objects have been outdated).

Most caches deployed for Web-based systems provide only weak consistency as the contents are relatively static. Furthermore, human users can make use of browser reloads to force the retrieval of the latest content. However, while there are more automated collaborative applications and program-driven agents used to retrieve and upload data on the Internet, the demand for strong consistency is increasing because these applications and agents are less tolerant of stale content. Maintaining strong consistency requires more frequent communication between the client and the server, which violates the principle of caching. Thus, using caching to reduce response time while guaranteeing the appropriate level of content consistency becomes a challenge for Web design, particularly in terms of caching architecture and algorithm.

### **2.5.2 Prefetching**

Traditionally, caching is a passive process where the content is only cached upon user request. However, it can be done proactively by means of a technique called Web cache prefetching where the content is cached before it is requested. The policy or algorithm adopted to determine the content to be cached greatly affects the efficiency of a prefetching Web cache. There are many different prefetching policies, for example, interactive prefetching prefetches a few documents referenced from the top of the requested document as well as the embedded objects required for these documents. Devillechaise *et al.* (2003) proposed a dynamic Web cache prefetching mechanism using Aspect-oriented

programming (AOP)<sup>2</sup> to dynamically weave and deweave prefetching policies. This allows dynamic installation of new policies in response to changing conditions. Devillechaise *et al.* showed that the overhead of dynamically weaving a new aspect was a few hundred microseconds but they didn't show if their prefetching mechanism has an improve cache hit rate.

Cohen and Kaplan (2000) suggested yet another prefetching method. Instead of prefetching the content, they proposed to prefetch the *means* for document transfer, such as DNS lookup and connection establishment. This includes resolving host names and establishing a TCP connection in advance, and sending a dummy HTTP head request to Web servers to “pre-warm” them. The underlying assumption for this technique is that DNS query times, TCP connection establishment, and start-of-session delays at HTTP servers are more important factors than pure transmission in leading to long response time. This finding was challenged by Marshak and Levy (2003) as the experiments on which the assumption was based were conducted in a LAN environment. In a LAN environment, the transmission bandwidth was not the bottleneck for system performance. Thus, the importance of DNS lookup and connection establishment in contributing to the overall response time will seem to be higher due to the faster data transmission speed. In the real world, since transmission bandwidth is typically lower than in a LAN environment, the users may suffer more from long delays due to transmission than from DNS lookup or connection establishment.

It is obvious that different prefetching algorithms will result in different content being prefetched into caches. The assumptions and heuristics on which prefetching algorithms are

---

<sup>2</sup> AOP is able to modify execution of a program at runtime by specifying the code to be modified (joint point) in the prespecified context (pointcut) and the functionality desired (advice).

based will greatly affect the effectiveness of prefetch caching in reducing Web page response time.

### **2.5.3 Content Distribution Network**

Content distribution network (CDN) is similar to caching but uses a different business model. For caching, content providers deploy caches (such as proxy servers) to give their downstream customers faster access to the content. The caches act both as clients and as servers. Content providers do not control the caches and the content is replicated as a function of user requests, except in the case of prefetched caching. CDN, on the other hand, is provided by third-party companies. Content providers are the customers of the companies. Customer data is replicated on the servers provided by the companies and the servers may contain data from many different customers.

From this perspective, CDN can be viewed as network based caching with a prefetching mechanism where content is distributed in advance to servers or caches located closer to the edges of the network. The servers or caches deliver content to the users on behalf of the content provider. CDN improves client-perceived response time by bringing the content closer to end users. It also reduces the need to invest in more powerful and yet expensive servers or more bandwidth in order to cope with an increasing user population as well as more demanding applications and Web content. Meanwhile, it also improves site availability by replicating static content in many distributed locations.

CDN is normally used to serve static content such as images or multimedia objects. However, the use of CDN techniques to serve dynamic data is increasing. Apart from the

content distribution servers, there are three key architecture elements required by CDN techniques (Iyengar *et al.*, 2002). They are:

- A distribution system that moves content from the origin servers into content distribution servers.
- An accounting/billing system that collects logs of client accesses and keeps track of content distributor server usage, primarily for network administration purpose.
- A request-routing system that directs client requests to appropriate servers. It may also interact with the distribution system to keep an up-to-date view of which content resides on which content distribution servers.

Similar to Web caching, CDN faces the familiar problem of the maintenance of consistency of the same content stored at different locations.

#### **2.5.4 Load Balancing**

Web servers need to handle many requests concurrently and therefore need to perform multithreading or multitasking in order to achieve parallelism. Additional parallelism can be achieved by using multiple servers in conjunction with a load balancer. The function of a load balancer is to distribute requests among the servers. One method of load balancing requests to servers is via DNS servers. DNS servers will translate, or resolve, the clients' requests into one of the associated IP addresses. A particular IP address will be chosen based on policies such as simple round robin, or server load information such as the number of requests received per unit time, as well as network geographic information.

One of the problems with load balancing using the DNS server is that name-to-IP mappings resulting from a DNS lookup may well be cached. The client requests could therefore bypass the DNS server logic and go directly to a server based on the cached information. This will cause load imbalance among the servers. To overcome this problem, the DNS server could assign a time-to-live (TTL) value to each name-to-IP mapping to indicate the duration after which the mapping becomes invalid. Using small TTL values can limit server load imbalance but increase DNS server load and response time, and vice-versa. Adaptive TTL algorithms, on the other hand, assign different TTL values for different clients. Smaller TTL values are assigned to frequently requesting clients compared to clients with low request rates.

Alternatively, a connection router can be used in front of several back-end servers. This type of connection routing hides the IP addresses of the back-end servers. IP addresses of individual servers won't be cached and this solves the problem caused by DNS load balancing. This approach also makes the addition and removal of the back-end servers transparent to the clients. The two approaches can be used in conjunction where a DNS server routes the requests to multiple connection routers to provide finer grained load balancing. Figure 2.9 gives a picture of this setup.

In order to assign client requests to different back-end servers, a load balancing policy is required. The round-robin algorithm distributes the requests evenly between the servers in a certain order. This algorithm is simple and easy to implement but does not take into consideration factors such as different servers having differing capability and different requests imposing different levels of load on the servers. As a result, it may lead to unbalanced loads. Weighted round-robin improves the algorithm over the deficiency. With



weighted round-robin, different weights are assigned to different servers based on their capability. Servers with higher weights will receive more requests than servers with lower weights.

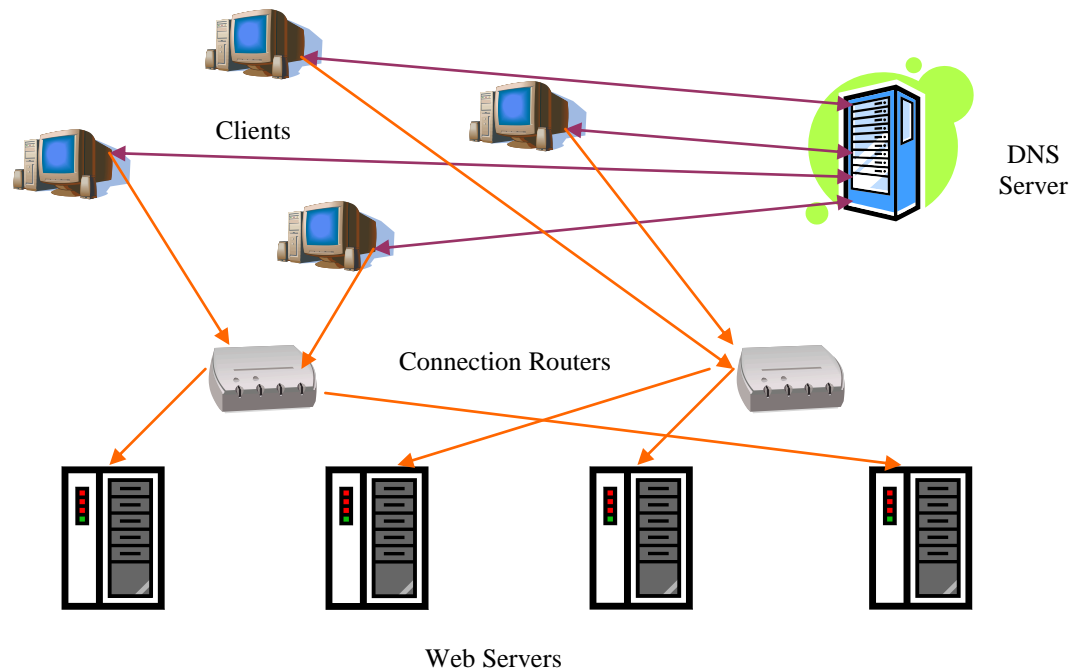


Figure 2.9 Load Balancing using DNS Server and Connection Routers

Connection routing can be done at level 4 or level 7 of the OSI model (see Figure 2.4). At level 4 (transport), the connection router knows nothing about the contents of the request. When done at level 7 (application), it is known as content-based routing. It allows more sophisticated routing techniques to be used. For example, dynamic requests could be sent to one set of servers and static requests to another set of servers. The major problem with content-based routing is the high overhead incurred.

Aweya *et al.* (2002) proposed a two-level approach which integrates admission control and load balancing. In their approach, incoming client requests are examined by Web switches placed in front of Web servers, interfacing with the network. If a client request relates to an

existing session between the client and the server, it is forwarded to the Web server concerned. Otherwise, the request is passed to an admission control function in the Web switch. The function determines whether there is sufficient capacity to service the new session, based on periodic load measurement provided by the Web servers. The server health status information, such as processor utilisation, available memory, queue lengths, response times, number of connections and sessions, can be obtained by the switches periodically polling the Web servers. If there is insufficient server capacity to process the request, the request can be rejected. Otherwise, it is passed to a load balancing or request distribution function. Based on the same periodic load measurements, the request distribution function determines which Web server will be assigned to handle the new session. This dual-step mechanism ensures that all requests accepted for processing by the Web servers are processed in an acceptable period of time.

Different load balancing schemes suit different workload or request patterns. For example, round-robin will be a cost saving yet effective method provided most client requests impose similar processing requirements on each of a group of servers with similar capabilities and handling a similar level of workload.

### **2.5.5 Decomposition of Dynamic Web Content**

Dynamic content generally takes longer than static content to download because the content is generated by the server on the fly before it is transmitted to the client. Static content can be retrieved from where it is stored and transmitted directly to the client without processing. Several attempts have been made to reduce response time for downloading Web pages with dynamic content. The common rationale shared among these attempts is to view dynamic

Web pages as quasi-static templates<sup>3</sup> that can be filled with cacheable and/or non-cacheable objects (Shi *et al.*, 2003). Shi *et al.* illustrated the concept using a personalised yet dynamic Web page as shown in Figure 2.10. S denotes an object which is shared while P denotes an object that is private.

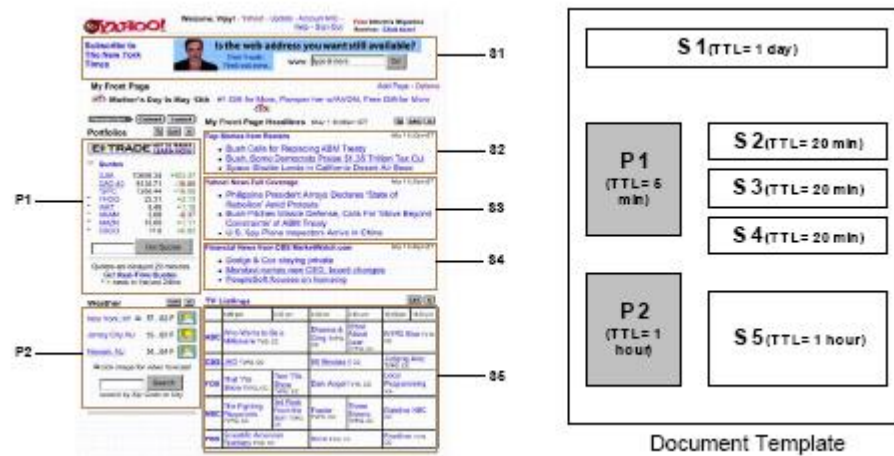


Figure 2.10 A Dynamic Page Viewed as a Quasi-static Document Template Filled with Objects Exhibit Different Cacheability Characteristics (Shi *et al.*, 2003)

Challenger *et al.* (2000) proposed a publishing system which supports the creation of dynamic Web page content. Web page content is viewed consisting of parts that can be automated and parts that must be proof read by humans. Complex Web pages are constructed from simpler fragments that may recursively embed other fragments. This view of dynamic Web content allows reuse of templates or cacheable fragments shared among different Web pages. Server load, bandwidth requirements, and response time can all be reduced in this way. A similar idea was proposed by Wills and Mikhailov (2000) where HTML pages are viewed as “*containers that hold distinct objects with heterogeneous type*

<sup>3</sup> A template expressed using formatting language such as Extensible Stylesheet Language-Formatting Objects (XSL-FO). XSL is a family of recommendations for defining Extensible Markup Language (XML) document transformation and presentation while XSL-FO is an XML vocabulary for specifying formatting semantics. (<http://www.w3.org/Style/XSL/>)

*and change characteristics”* (p. 1). HTML requests and responses are interpreted and handled as relationships between containers and embedded objects.

One of the problems with decomposing Web content into smaller and more reusable fragments or components is the identification of components that make up Web pages and to differentiate between cacheable and non-cacheable components. The relationships between the components and Web pages could be difficult to identify and this causes difficulty for Web page designers.

### **2.5.6 Server Tuning**

There are some configuration parameters related to Web servers that can be tuned in order to improve the performance of the server and consequently impact on response time. Apache Web server, for example, has parameters like MaxClients and KeepAlive which may be adjusted to affect server performance metrics like CPU and memory utilisation. Instead of tuning these parameters manually, Diao *et al.* (2003) from IBM proposed a mechanism to automatically tune MaxClients and KeepAlive parameters for the Apache Web server at run-time in order to achieve the desired CPU and memory utilisation. This relieves the system administrator from the tedious, time-consuming, error-prone, and skill-intensive manual adjustment of the parameters. MaxClients defines the maximum number of worker processes that are responsible for communicating with the clients. A higher MaxClients value therefore allows the Apache Web server to process more client requests concurrently and results in higher CPU and memory utilisation. KeepAlive specifies the maximum allowed time for a persistent connection between the client and the server to remain open. Decreasing the value of KeepAlive makes the worker processes more active and thus implies higher CPU and memory utilisation. This may seem to be not directly

related to reducing response time, but it does help to keep the Web server at an optimum load level. Referring to Figure 2.8, it can be seen that keeping Web server load under a threshold prevents response time from increasing exponentially, which would happen if the load exceeded the threshold.

### **2.5.7 Customisation**

Web content can be customised to suit the needs and capability of different clients or based on the network traffic condition. Steinberg and Pasquale (2002) used dynamically deployable software modules called Web Stream Customizers (WSCs) located between Web clients and servers to adapt Web content as necessary to improve performance, including response time, reliability and security of the Web service. Their Web middleware architecture is especially useful for client machines with limited resources such as wireless palmtops. WSCs provide adaptive system-based and content-based customisation to achieve its goals. The adaptive customisation includes removing data that users are not interested in, filtering images to smaller representations, and displaying Web pages in an easy-to-surf format. Web Content may be compressed or encrypted before it is transmitted onto wireless link which generally has lower bandwidth, and then decompressed or decrypted at the client if the client is powerful enough to deal with this. Compression and encryption can even be an optional step depending on system conditions or user behaviour. For example, compression and encryption may be done only when network throughput is low enough or until the overhead of compression and encryption outweigh delay caused by network congestion.

Such Web middleware architecture not only supports content customisation dynamically, but also supports existing Web servers, clients, and structures seamlessly, without the need

to modify them, in order to achieve better response time. The overhead of the customisation is also low compared to the real Web transaction times (Steinberg & Pasquale, 2002).

### **2.5.8 Web Standards**

Web standards are formal technical specifications that define and describe different aspects of the Web. Web standards ensure that Web-based content can be created and interpreted correctly by different browsers as well as ensuring the content is accessible via different platforms and devices. Some Web standards are also relevant to response time. One such standard is HTTP/1.1. Prior to HTTP/1.1, each request made by the client required a TCP connection to be established between the client and the server that served the request. Since a Web page will typically consist of an HTML file as the container and a few embedded objects, such as images, downloading a Web page would involve the client making more than one request to the server and each request would involve establishing a TCP connection. According to Chandranmenon and Varghese (2001), setting up such a connection may take up to 80ms.

With HTTP/1.1, persistent connection and pipelining are supported (Fielding *et al.*, 1999). Persistent connection allows a TCP connection to be re-used to retrieve multiple objects from the same IP address. Pipelining, on the other hand, allows the client to make a series of requests over the same connection without waiting for the completion of the previous response. Both persistent connection and pipelining will reduce response time for downloading Web pages. Studies from Cherkasova *et al.*(2003) and Krishnamurthy and Wills (2000) show that HTTP/1.1 does have a positive effect on reducing response time. Apart from using HTTP/1.1 to improve performance, it is also possible to use HTTP/1.0 with a KeepAlive parameter which specifies the lifespan of a TCP connection to gain

similar benefit as using HTTP/1.1. However, if a Web page can be downloaded through a single client request, such as for example, a plain HTML page, then HTTP/1.1 will have no advantage over HTTP/1.0.

### **2.5.9 Web Technologies (Software)**

Many software technologies are involved in different aspects of the Web, including client and server operating systems, Web browsers, server software, databases, middleware architectures, and scripting engines and languages. There are some relevant performance tips given by (Killelea, 2002):

- The Web browser seldom becomes the bottleneck that causes lengthy response times but its settings may well be tuned to achieve slightly better performance. For example, not verifying cached pages from the server makes retrieving of the cached pages faster even though it risks the user viewing out of date pages.
- On identical PC hardware, Linux generally gives better performance as a Web client than Windows.
- Unix is more stable and has better performance than other server operating systems because of its longer development history and open nature.
- To improve database performance, actions can be taken include to use precompiled SQL statements called *prepared statements*, cache the results of the most frequently used queries, and use a connection pool rather than setting up a connection for each database query.

Since dynamic Web content has become an essential component of many Web sites nowadays, different aspects related to dynamic Web content are studied. One of the aspects

is technologies for generating dynamic Web content. Cecchet *et al* (2003) studied three middleware architectures used for generating dynamic Web content, namely Hypertext Preprocessor (PHP), Java servlets, and Enterprise Java Beans (EJB). The architectures use different mechanisms for generating dynamic Web content.

- PHP scripts are tied to the Web server and require the writing of explicit database queries.
- Java servlets execute in a different process from the Web server, i.e. the Java Virtual Machine (JVM). This allows Java servlets to be located on a different machine from the Web server for better load balancing. Database queries are written explicitly, but the Java synchronisation primitives can aid in the communications with the database.
- EJB is a server side component that is used to abstract application business logic from the underlying middleware. An EJB server manages several EJB containers that in turn manage enterprise beans contained within them. The enterprise beans provide different yet common services such as database access, transaction management, messaging, and naming services.

The three architectures were tested on two common application benchmarks: an online bookstore and an auction site. Even though measurements are made in terms of server throughput in number of interactions per minute, and the percentage of server CPU utilisation, the measurements do give some indications of how well these different middleware architectures are at handling client requests and how response time might be affected. As the number of client requests increases, server load will increase too, and so do throughput and response time, as can be seen from Figure 2.8 (see page 30).



Overall, PHP scripts are the most efficient of the three architectures. However, PHP scripts provide limited yet insecure functionality and runtime support. Java servlets can be offloaded to another machine for better performance if the Web server is the bottleneck. Servlets also help to resolve the database lock contention problem<sup>4</sup> if they are the only application that accesses the database. EJB offers the most flexible architecture and supports good software engineering qualities such as modularity, portability, and maintainability. Unfortunately the performance of EJB is the worst among the three architectures.

Titchkosky *et al.* (2003) did similar research to the one outlined above. They compared the performance of Perl, PHP, and server-side Java. They measured performance for serving both static and dynamic content. For serving dynamic content, both cases, with and without database access, were examined. However, as compared to the above research, the workload used for the measurements was more general so that it would suit any site using dynamic Web content generation, rather than being tailored to the interests of online bookstores or online auctions. According to the study, PHP handles *small* (2kB response size) dynamic content requests well but does not cope with *large* (64kB response size) dynamic content requests well. Java servlet containers are weak at serving static content, but perform better for serving dynamic content.

#### **2.5.10 Web Page Content: Composition, Generation and Structure**

Web page content plays a major role in determining response time. For example, whether a Web page comprises dynamic or static content and the extent of database accesses is

---

<sup>4</sup> A lock is a synchronisation mechanism for enforcing limits on access to a resource to prevent multiple users from making conflicting modifications to a set of data simultaneously. Lock contention occurs whenever one user (a process or thread) attempts to acquire a lock held by another user.

involved in generating the Web page, will affect response time. Table 2.2 below is an excerpt from Titchkosky *et al.*'s study (Titchkosky *et al.*, 2003). It shows the peak rate supported by Apache 1.3.27 Web server in producing different types of responses in handling client requests, with identical hardware and software settings.

Table 2.2 Peak Rate Supported by Apache 1.3.27 Web Server in Producing Different Types of Responses

Type of Response	Peak Rate (Responses/Second)
2kB Static File	4, 000
64kB Static File	1, 400
2kB Dynamic (PHP) File without Database Access	1, 400
64kB Dynamic (PHP) File without Database Access	250
2kB Dynamic (PHP) File with Database Access	850
64kB Dynamic (PHP) File with Database Access	250

They found that due to the overhead for database access and server processing for dynamic content, the peak request rate supported by a server can be reduced up to eight times, depending on the workload characteristics and the middleware technologies used. Supporting higher peak request implies a server is less likely to have an exponentially increasing response time as the number of client requests increases.

With more demanding Web applications and services in place nowadays and more Internet users with increased Internet activities, it is not surprising that dynamic Web content makes up a bigger portion of the Internet traffic than ever before. The portion is increasing daily. With the constraint of the existing network bandwidth and increasing number of users, response time is therefore becoming an ever more important QoS issue.

Another related issue between Web page content and response time is hyperlink structure in the page. The organisation of Web pages affects the time taken by the users to reach their

desired Web pages within the site. If the page can be reached with fewer hyperlink clicks, it does not only imply shorter “overall latency”, as perceived by the client, but also implies fewer “transactions” taking place between the client and the server. This will reduce the server workload and network traffic eventually.

Garofalakis *et al.* (1999) proposed an automated method to reorganise Web page hyperlink structure based on page popularity. Page popularity was defined in terms of relative accesses instead of absolute accesses to the page. Relative access is a function of absolute access, proportional to the depth of the page in the hyperlink structure and number of the pages at the same depth; and inversely proportional to the number of references (hyperlinks) to the page from other pages of the Web site. The study found that rearranging a site’s hyperlink structure without changing the pages’ content could increase average absolute access to the pages of the site, the mean number of pages accessed by each user, and the average time each user spent per session on the server. The results showed that, on average, the users spent more time and visited more pages at the Web site. However, it does not imply that the users are able to reach their desired pages faster. In fact, the users might have stayed longer at the Web site due to difficulty in finding the information they wanted.

Czyzowicz *et al.* (2003) also dealt with hyperlink structure. They introduced an algorithm to assign added hyperlinks called hotlinks (or in other words, shortcuts) to most popular Web pages from their “ancestor” pages. The outcome of the study was a proposal to suggest to the Web page designer which hotlink should be added to which page. However, there is no evaluation on whether or not the proposal improves response time.

One of the problems with reorganising hyperlink structure or adding shortcuts to most popular Web pages in other pages is that it may break the logical structure of a Web site even though frequently accessed Web pages are brought closer to the user.

### **2.5.11 Hardware**

Hardware forms the structure on which the Web operates. Hardware making up the client, server and network has an impact on response time. On the client and server sides, hardware aspects that are commonly related to response time include the processing power of the central processing unit (CPU), the capacity and speed of the memory (random access memory – RAM), cache, bus, and disk. Web client's CPU is less important than input-output (I/O) devices in determining response time as Web surfing is an I/O-bound activity rather than a CPU-bound activity (Killelea, 2002). Nevertheless, the CPU speed does greatly affect the efficiency of HTML and image rendering as HTML file parsing creates a significant CPU load.

On the server side, hardware is a more complicated issue. The server needs to handle many client requests simultaneously and generate dynamic content, so the CPU becomes a more important component than it is on the client. This is especially true when there are many clients making requests at the same time. However, it is still worthwhile to remember that *“a Web server is essentially a remote storage that copies data from its RAM or disk to the network connection upon request”* (Killelea, 2002, p. 264). Therefore, high performance I/O devices are crucial to a good Web server, which is highlighted by, among other things, fast response time. Indicators to high performance I/O devices include having high speed disks and multiple dedicated busses for cache, I/O, RAM and peripherals to reduce contention for the busses. Having huge memory capacity also reduces disk access and

improves I/O speeds. On top of that, a server requires a network interface card (NIC) that provides the connection between the network cable and the server's bus. NICs have buffers to hold and transmit data between the server and the Internet. How quickly an NIC can get and send data from the RAM or disk to the network and how efficient the CPU, RAM and bus are at handling interrupts from the NIC and placing its data from the network on to the server's bus, affects response time.

The "network" refers to the medium that connects the client and the server in this context. The medium is made up of lines (different kinds of cables and transmission medium) and terminators or connection points (modems, routers, repeaters, bridges, hubs, and switches). A pair of terminators and a line comprise a connection segment on the Internet. The connection segments form the network to carry and forward data packets from their source to destination. There is inevitable latency in the process of carrying and forwarding data packets, imposed by physical properties of the line and line protocols deployed at the terminators. The theoretical minimum boundary of latency every physical line has is the speed of light but the number of terminators on the Internet data packets have to pass through is a more significant factor that causes delay (Killelea, 2002). The terminators control the speed at which data packets can pass through. The latency caused by the terminator will be more noticeable if it has to perform protocol conversion for incoming and outgoing data.

Upgrading hardware to improve system performance, including response time, seems to be straightforward especially in the eyes of individual user, but it may be expensive. For example, C2C, a 10, 500-mile undersea cable built in 2002 that links China, Hong Kong,

Japan, South Korea, the Philippines, Singapore and Taiwan to the United States cost two billion US dollars (Greenlees & Arnold, 2006).

## **2.6 Measuring Response Time**

Response time measurement methods for a Web site can be grouped into three categories:

- Active probing,
- Server-side measurement, and
- Client-side measurement.

The three categories are described in detail below.

### **2.6.1 Active Probing**

With active probing, a few geographically distributed synthetic clients, called the agents, are used to periodically probe the server by requesting a set of Web pages or operations. The agents mimic users from different locations over the world. The measurements obtained are representations of latencies that may be experienced by the end users. Active probing therefore produces real world results from faked end users. In order to produce results representative enough of the massive Internet traffic from largely diversified users across the world with varied connection speeds, active probing requires machines with different capabilities to be setup in many different locations and large amounts of measurement to be taken daily. One of the examples of active probing is Keynote (<http://www.keynote.com>), a commercial provider of test and measurement products for Internet performance. As of October 2006, Keynote has 2, 400 measurement computers and mobile devices in over 240 locations and 160 metropolitan areas worldwide. More than 100,

000, 000 Internet performance measurements were taken daily. Figure 2.11 presents the idea of active probing.

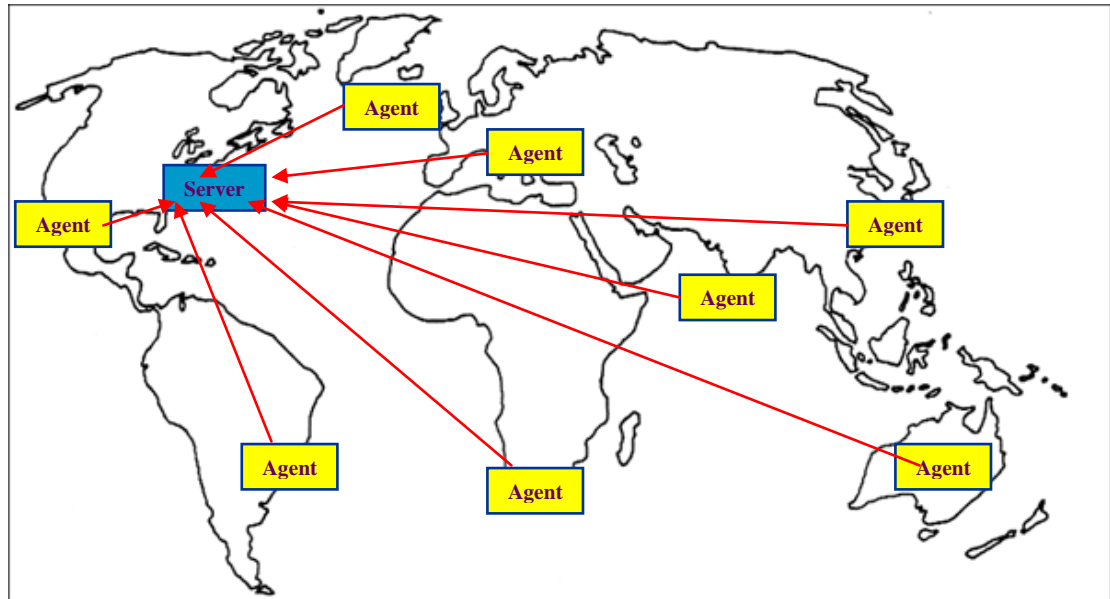


Figure 2.11 Active Probing: Agents Distributed across the World Actively Probing the Server to Perform Measurements

### 2.6.2 Server-side Measurements

As the name implies, server-side measurement performs measurements of various performance metrics at the Web server. Server log analysis is the most commonly used method. A Web server log contains fields that describe each request a browser makes from the server (Rosenstein, 2000). The fields include:

- The IP address or the host name from which the request originated.
- The date and time of when the request was made.
- The request line that shows the method by which the resource was requested (such as GET or POST), the resource requested together with the protocol used by the client (such as HTTP/1.0 or HTTP/1.1).

- The status code that indicates if the request resulted in a successful response (codes beginning in 2), a redirection to other resource (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5).
- The size of the object returned to the client excluding the response headers.
- The browser or user agent field that indicates the browser used by the user to make the request.
- The referer field that indicates the page from which the user made the request.

Different information can be extracted from these fields to fulfil the interests of different parties. For example, the “IP address” field can be used to show the geographic distribution of visitors to the Web site while the “user agent” field tells which browser was used by most of the visitors. Even though the server log could provide a large amount of data with rich attributes that can be transformed into appropriate and easily assessable actions (Kohavi, 2001), it nevertheless has difficulty to produce desirable information from log files.

Since the server log only records individual requests and downloading a Web page may yield a few non-consecutive entries in the log, it does not provide direct information about the time taken to download a full page, unless heuristics are used to identify sessions from the log entries. Furthermore, if a number of clients are situated behind the same proxy server, they will appear to have the same IP address (of the proxy server) in the server log. Apart from that, if the client requests and retrieves Web content that is cached, the requests won't appear in the server log.



Despite the above-mentioned problems, server log analysis is still a favourite method for performing measurements, probably because of its availability. One of the reasons is that it doesn't introduce traffic into the network (Ardaiz *et al.*, 2001). Other advantages include the fact that it can evaluate each client's experience without the need for instrumentation at every client or additional agents placed on the Internet, and implementation at the HTTP level rather than lower protocol levels (Marshak & Levy, 2003). Server side measurements also reduce concerns about intruding on users' privacy, which is a problem with instrumented page.

To overcome the problem related to the absence of session information in the server log, Krishnamurthy and Wills (2002) proposed the following heuristics to extract information from the server log and induce a sequence of requests that make up the download of a Web page:

- The first request by a client, indicated by the client IP address or host name, returns an HTML object or the output of a server side script, called the base object.
- Each subsequent request from the client is for an image, style sheet, or JavaScript object.
- If the referer field is set, then the referer field for the embedded objects must also match the URI of the base object.
- An arbitrary maximum threshold of 60 seconds is defined between the time the base object was downloaded and any subsequent requests. Any subsequent requests made beyond this threshold are not classified as making up the original Web page. The value of this threshold is a significant factor that determines the accuracy of the approximation of response time.

Another approach used by Marshak and Levy (2003) was to add a tiny inline HTTP object called the sentry (better known as a Web bug) at the end of the HTML document. They assumed that the request for the sentry will be the last corresponding entry recorded in the server log for a Web page downloaded. However, this assumption is flawed because HTTP does not require embedded objects to be requested in the sequence as they appear in the HTML document. Thus, the sentry is not necessarily the last object to be requested even though it is placed at the end of the document.

Figure 2.12 shows the simplified architecture about how server log analysis is used to estimate response time.

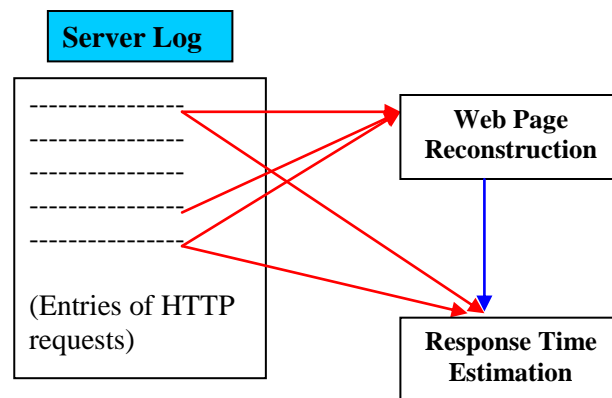


Figure 2.12 Using Server Log Analysis to Estimate Response Time

Besides server log analysis, server-side measurement methods also include measurements made at lower protocol levels. For example, Cherkasova *et al.* (2003) proposed a tool called EtE monitor to measure response time by passively collecting packet traces from a server site. The EtE monitor architecture is shown in Figure 2.13.

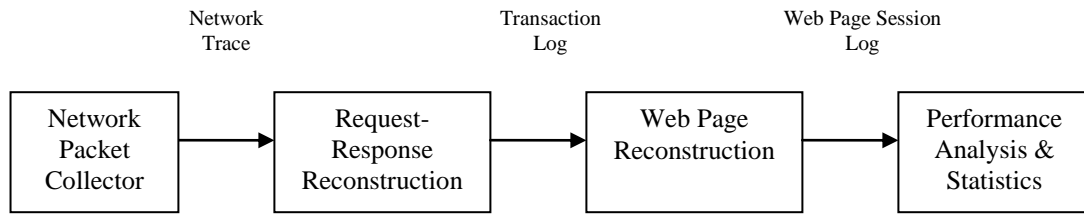


Figure 2.13 EtE Monitor Architecture (Cherkasova *et al.*, 2003)

The network packet collector module collects network packets using tcp-dump<sup>5</sup> and records them to a network trace for offline analysis. The request-response reconstruction module uses the network trace to reconstruct all TCP connections using client IP addresses, client port numbers, and request-response TCP sequence numbers. HTTP transactions are then extracted and the HTTP header lines of each request recorded in the transaction log for future processing. The Web page reconstruction module groups underlying physical object retrievals together into logical Web pages and stores them in the Web page session log. The Web page reconstruction process is similar to that achieved by extracting information from the server log to identify requests that make up a Web page. Analysis thereafter could be done based on the reconstructed Web page. Again, the accuracy of the reconstructed Web page relies on the heuristics used and the amount of data available for inferring Web pages from raw data.

### 2.6.3 Client-side Measurements

Client-side measurements make use of instrumentations such as scripting languages or specialised software at the client side to acquire the desired information. Cookies might be used to record information at the client side. A cookie is a small parcel of information

---

<sup>5</sup> A computer network debugging tool that allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached.

issued by the Web server to a Web browser to uniquely and anonymously identify the user using that particular browser (Clickstream, 2003).

Rajamony and Elnozahy (2001) used JavaScript to instrument hyperlinks in a set of Web pages for measuring response time. When an instrumented link is activated, the current time is determined and remembered, and then the request is sent to the Web server. After the requested page and its embedded elements have been fully loaded to the browser, the client browser computes the response time as the difference between the current time and the previously stored time. The response time can be transmitted to a record-keeping Web site on a different server from the originally responding Web server for further analysis. Note has to be taken that the time samples taken need to be stored in cookies, a dedicated window or a frame within a browser window as the browser cannot maintain the values across page loads any other way. Figure 2.14 depicts this approach, which allows accurate measurement of response times as experienced by the end user.

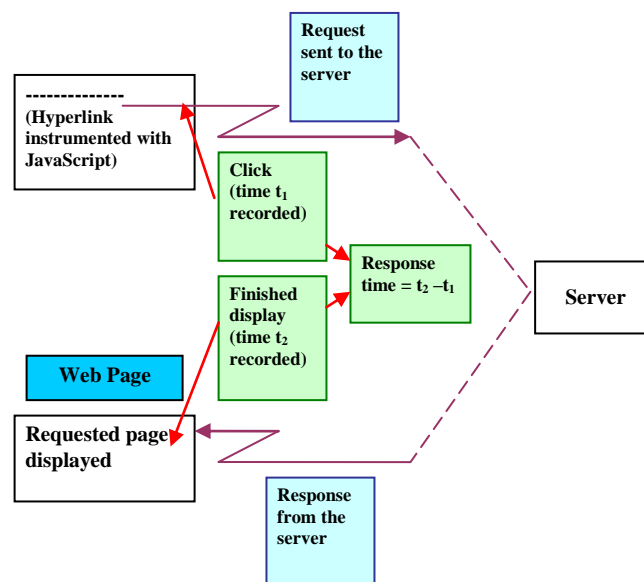


Figure 2.14 Client Side Measurement of Response Time using JavaScript Instrumented Links

There are however, four problems with this approach. Firstly, the approach does not compute response time for a request which is not made through an instrumented hyperlink. Secondly, pages containing images or PDF files cannot be instrumented. Thirdly, the approach does not work with browsers that do not support the scripting language used for instrumentation, or when the execution of the scripting language is disabled by the user. Fourthly, the cookie itself impacts response time.

Clickstream (2003), a company that *collects, transforms* and *transmits* information on its customer and commercial activities on Web sites or any other HTTP environment (<http://www.clickstream.com>), uses third party cookies for measuring response time as well as page view time and tracking user activities when surfing the monitored pages. Cookies are used to deal with the problem of HTTP being a stateless protocol. Cookies are able to keep track of user sessions. Thus, communication between the client and the server, as well as the user activities on the browser, can be detected and recorded by the cookies. Clickstream installs a software module on the master content server. The software module embeds tracking code into the header of each Web page dispatched by the server. The module also manages and processes tracking data that is returned to the server as a tracking cookie from the Web browser. Even though Clickstream claims to be a non-intrusive and privacy friendly system, as long as cookies are used, the concern about the line between acceptable monitoring and intrusion of user privacy will always exist. Moreover, this method only works when cookies are enabled at the user's browser.

## **2.7 Conclusion**

In this chapter, the historical and technical backgrounds of the Internet and the Web have been described. Performance issues pertinent to the Internet and the Web were discussed in

general terms. Among the performance issues, response time was explained in particular to give a deeper understanding of the issue and form a foundation for discussions in later chapters. Ways to improve response time and methods to measure it were also discussed. The next chapter will focus on the problems to be addressed in this research, as well as the framework and methods for addressing the problems.

## **CHAPTER 3**

### **PROBLEM STATEMENT**

Background knowledge on the Internet, the Web, computer system performance, and response time were presented in the previous chapter. This chapter continues the discussion by outlining the specific problem addressed by this researcher, i.e. Web page response time. The methods used to conduct the research are also explained in this chapter. They are presented as follows:

- Section 3.1: Background of the problems to be addressed.
- Section 3.2: Research framework and approach.
- Section 3.2: Conclusion

#### **3.1 Background and the Problems**

My research is concentrated on the area of response time measurement, analysis and improvement of Web-based systems. The initial scope of this research is, in a broad sense, of performance for computer systems. The interest was motivated by Leung (1988) and Lilja (2000) who provide an overall background knowledge for the field of interest. Haban and Wybraniec (1990)'s study focused on performance for distributed systems and inspired my more specific interest in performance of Web-based systems. A review of relevant literature was presented in Chapter 2.

Based on the literature review, it is clear that most researchers study Web-based systems' performance from an overall system perspective. It is also found that the virtually unbound and unpredictable usage patterns of Web-based systems make evaluation of their

performance challenging. Many existing studies focus on measuring how well Web servers handle user requests (Aardt, 2002; Cecchet *et al.*, 2002; Mosberger & Jin, 1998; Nahum *et al.*, 2001; Savoia, 2001; Smith, 2001; Strahl, 2000) and determine how to modify the configuration (Diao *et al.*, 2003) of the servers, where necessary, to enhance performance. Some researchers do examine the characteristics of Web pages, but not in relation to the system's performance. Instead, Web pages are only studied from the perspective of user interface design and usability (Chadwick-Dias *et al.*, 2003; Geissler *et al.*, 2001; Mandel & Johnson, 2002; Nielsen, 2000; Ward & Marsden, 2003), information management and retrieval (Germonprez & Zigurs, 2003; Larson & Czerwinski, 1998; Song *et al.*, 2004), and marketing or e-commerce (Al-Diri *et al.*, 2006; Debaraj *et al.*, 2003). Other researchers concentrate on transmission related issues (Ariga *et al.*, 2000; Fujinoki *et al.*, 2003; Mohamed *et al.*, 2006), which are beyond the scope of this research.

Very few researchers, if any, perceive Web pages to be individual entities, the content of which may well influence page delivery performance. In fact, a Web-based system is usually considered to be a structure that consists of clients, communication networks, and servers. The relationship between Web page and system performance, particularly its response time, has not yet been studied thoroughly. Anecdotal statements had been made about the relationship between Web page and its response time, e.g. “... *speed must be the overriding design criterion. To keep page sizes small, graphics should be kept to a minimum ...*” (Nielsen, 1997) and “*It is recommended to design ‘slim’ pages. To this end, we should limit the number of images and other high-volume media elements per page...*” (Hitz *et al.*, 2006: p. 225). There is, as yet, no systematic scientific research that studies a Web page in association with its response time. My research has filled the gap by investigating this aspect of Web-based system performance.



## **3.2 Research Framework and Approach**

Web page response time is a quantitative performance metric and thus measurement of its values is the most straightforward yet essential way to deal with it. Measurement produces a set of values that show what the response time of a Web page is at a specific time. However, merely relying on measurement is insufficient to provide a better understanding of response time, or to gain a sense of the degradation that could occur based on changes in the system as it runs from day to day. Modelling can therefore be used to present response time at an abstract level. My research proposes models that explain how response time is related to particular Web page characteristics. Once the Web developer is satisfied with the Web page, both in terms of functionality and estimated response time (amongst other non-functional requirements), the Web page can be released and be used as part of a functioning system. At this time, a practical way to observe ongoing response times of specific in-use Web pages is desirable and necessary. Continuous monitoring can be carried out to accomplish this purpose of observing response time over a period of time. Measurement, modelling, and monitoring form the core modules of this research, which aim to support the improvement of Web page response time during development and maintenance phases.

During the Web page development phase, the page developer would like to have an idea of the expected response times of individual pages before they are published to the Web site. The developer can use the measurement module proposed in this research to examine response times of the individual pages. Before measuring the response times, the developer can first assign a maximal response time for the Web site. This could be prescribed by the service level agreement entered into between the developers and the customer, or could be based on knowledge of the maximum time a user is willing to wait for a page. Alternatively,

different maximal response times can be assigned to different pages based on their importance, criticality, and complexity of the functionality encompassed within the page. After the measurement process, the maximal response times can be compared with the measured response times. Web pages that exceed the assigned maximal response times can then be identified. Remedial action can be taken to modify the Web pages so that their response times can be improved.

In order to modify the Web pages to improve their response times, a Web page designer or developer will need to understand and identify the individual characteristics of Web pages that influence and impact on their response times. Models that explain the relationship between Web page characteristics and response time are useful in this context. The Web page designer or developer can check the Web pages against the models to identify any design deficiencies that are known to lead to poor response times. Modification efforts can then be focused on those specific aspects to improve the response times. Models assist the developer in spending time on those aspects which do indeed impact on response time, and not to waste time blindly trying all sorts of modifications in the hope of improving response time.

Even though measurement during the Web page development phase helps to reduce poorly designed Web pages in terms of response time, it does not guarantee satisfactory Web page response time during the day to day operation of the Web site. During daily operation, the Web pages may still exhibit slow response time due to certain other factors. Sometimes Web pages which perform very well at the launch time will start developing poor response time as the system matures. The monitoring module can be used to identify these Web pages. If a Web page consistently exhibits slow response time, regardless of factors such as

the number of concurrent users accessing the Web site, it may indicate that the Web page needs to be investigated and modified to achieve better response time. At this stage, the Web pages can again be checked against the response time models to identify further areas for improvement.

From the descriptions above, it is clear that the Web page designer, developer, and maintainer are interested parties to the measurement, modelling, and monitoring modules proposed in this research. As a tool, the package presented here can be used to identify problems, delineate realistic response time expectations, highlight poorly responding pages and support effective remedial action.

Figure 3.1 illustrates the framework of this research. It shows that the research consists of the three modules, namely, measurement, modelling, and monitoring, which encircle around Web page response time. The modules are explained briefly below.

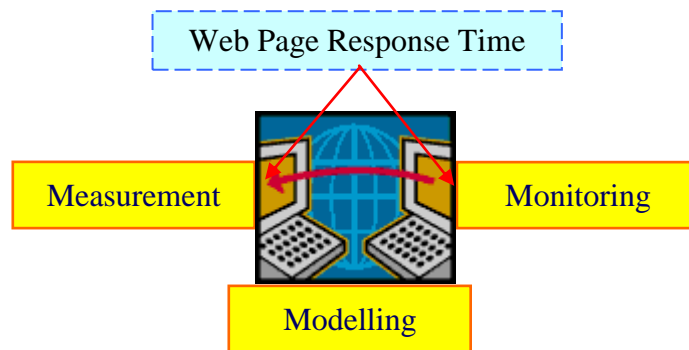


Figure 3.1 Research Framework

- Measurement. This module uses a combination of client and server-side measurements to measure response time and decomposes the time into three of its

constituents: generation, transmission and rendering. Two measurement tools were developed to provide the Web page developer with a means for checking the response time. The module is introduced in Chapter 4.

- **Modelling.** Various Web page characteristics that affect its response time are identified. Models of Web page response time as functions of these page characteristics are developed. The models support the Web page developer to check design quality of the Web pages in terms of response time. The models are verified using the measurement tools developed. It is detailed in Chapter 5.
- **Monitoring.** As compared to the measurement module, the monitoring module provides an alternative way to estimate Web page response time. Monitoring will be done at the server side, relying on server log analysis. Monitoring helps the operational team to observe response times of the Web pages and the maintenance team to identify Web pages for modification to improve response time. A tool was developed to demonstrate the monitoring activity. Chapter 6 gives a complete explanation of this module.

Meanwhile, approaches used to conduct the research include:

- **Laboratory experiments.** In order to minimise the impact of various factors other than Web page characteristics on Web page response time, especially network conditions, experiments were conducted in a controlled environment, i.e. a LAN. Isolating the effects of these factors from Web page characteristics as far as possible is important to identify the significance of Web page characteristics in affecting response time. An individual server was set up to examine the impact of different

Web page characteristics on response time. This eliminated possible effects of other user requests.

- Mathematical analysis. Empirical data obtained from experiments were analysed to find out relationships between Web page characteristics and response time. Models were developed to describe these relationships in a clear yet simple way.
- Software development. Different software tools were developed for data collection, performance monitoring and analysis. Software approach is chosen rather than hardware approach for its flexibility and concern of costs.

### 3.3 Conclusion

This chapter introduces the problems to be addressed by this research and the methodology used to address the problems. The essence of the problems to be addressed is indeed Web page response time. The research framework proposed in this chapter to solve the problems consists of three modules: *measurement*, *modelling*, and *monitoring*. Measurement and modelling are particularly useful to Web page designers and developers to ensure the Web pages do not exhibit slow response time due to the pages' characteristics before they are published. Monitoring, on the other hand, is useful for the operation team.

Measurement is discussed in the next chapter which describes methods for measuring Web page response time and decomposing the time into three of its constituents, i.e. *generation*, *transmission*, and *rendering*. Data obtained from measurements of a prototype Web site are also presented and discussed in the next chapter. Modelling is discussed in Chapter 5 while Chapter 6 deals with monitoring. Chapter 7 explains how the three modules can be used to enhance Web page quality in terms of response time.

## CHAPTER 4

### MEASURING AND DECOMPOSING WEB PAGE RESPONSE TIME

This chapter focuses on two methods used to measure response time for downloading a Web page and explains how the methods are used to decompose the response time into three of its constituents: *generation*, *transmission*, and *rendering*. Section 4.1 describes components of Web page response time. Sections 4.2 and 4.3 introduce two Web page response time measurement methods. Examples of the use of the methods to analyse Web page response time for a pair of synthetic Web client and server are presented in Section 4.4. The results are discussed in Section 4.5. Section 4.6 concludes the chapter.

#### 4.1 Components of Web Page Response Time

From a high-level view, retrieving a Web page involves three parties: the client, the server, and the network, as depicted in Figure 4.1.

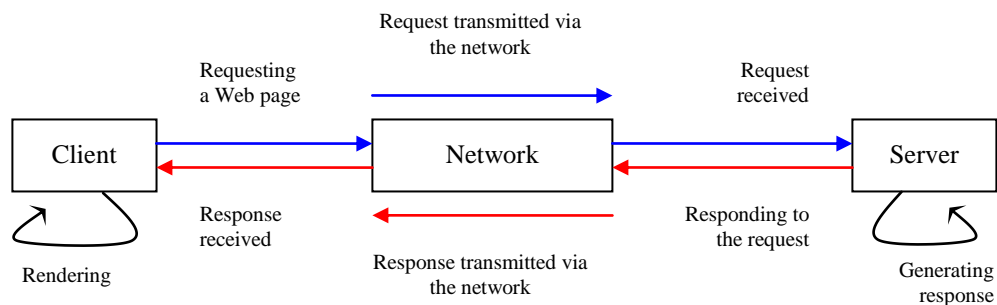


Figure 4.1 Three Parties Involved in Retrieving a Web Page

The client is responsible for initiating the request and rendering the server's response on, typically, a Web browser. The server is in charge of processing the client's request and

responds to the request accordingly. The network resides between the client and the server and ensures that the request and response are transmitted to their destinations correctly. Thus, the whole process consists of three major activities taking place:

- On the client: Rendering of the server's response.
- On the network: Transmitting the client's request and the server's response.
- On the server: Generating the response to the client's request.

Each of these activities consumes time that makes up the eventual response time experienced by the end-user. Measurements of the three parts involved can be done at different points to determine time taken for different activities, or to measure overall response time as a single quality. For example, the client browser can be instrumented to record the time a request is made and when it completes display of the requested page. The difference between these two time instances denotes total response time of the Web page.

Such a method is useful to determine whether users are experiencing long delays when downloading a page. However, it does not tell us about possible reasons for the delay. A new method has therefore been designed to overcome this deficiency. The method not only measures response time as experienced by end-users, but also captures other information that facilitates further analysis of the response time. The method may well introduce notable measurement overhead as a trade-off for providing more details about Web page response time. Thus another method, which has lower measurement overhead but provides less response time details, has been designed as an alternative for the measurement of response time. The first method relies on a proxy server while the second uses a Web page with two frames.

## 4.2 The First Method: Proxy Server

The method is a combination of client and server-side measurement. It measures response time and decomposes the time into three of its constituents: transmission, generation, and display. In order to do so, the following components are required:

- A Muffin<sup>6</sup> (<http://muffin.doit.org>) proxy server is configured and placed between the client and server to record time elapsed between the request for an HTML page (and its embedded objects) and the receipt of the page and embedded objects.
- An extended server access log records server processing time for each client request.
- Web pages that are instrumented using JavaScript to record the time taken to render the pages into a Web browser.

Figure 4.2 illustrates the method.

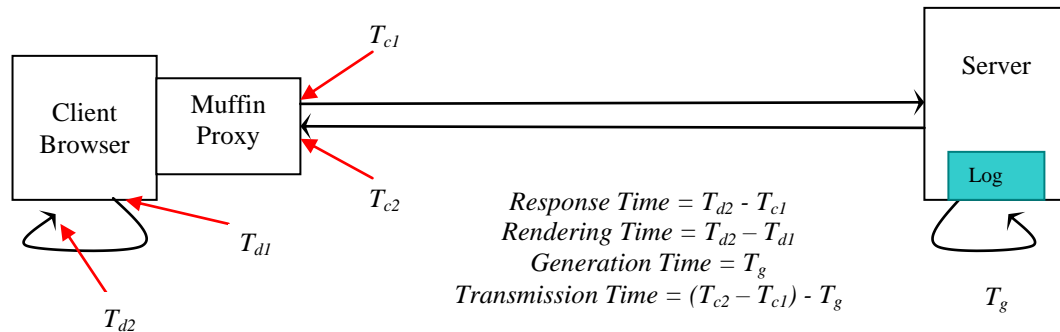


Figure 4.2 Measuring and Decomposing Response Time Using Muffin Proxy Server

When the client requests a Web page, the request will be parsed and the time recorded by the Muffin proxy server. The request is then forwarded to the server. After processing the request, the server will return the required response to the client. Information about this matching request-response pair will also be recorded in the server's access log. The format

<sup>6</sup> Muffin is a WWW filtering system written in Java. The name Muffin implies "a big (HTTP) cookie".



of the log file includes one of the W3C extended log file fields, the *Time Taken* field. The field records the time taken, in milliseconds, by the server to process the request.

The server's response will be parsed and the time the response was received will be recorded by the Muffin proxy server before the client browser renders it. All embedded objects will go through these steps. Thus, the times when each object was requested and received will also be recorded by the Muffin proxy server.

The response has been instrumented using JavaScript so that when the browser starts and completes display of the Web page, both times are recorded. Figure 4.3 shows the JavaScript code to measure and display this page rendering time, invoked by the *onLoad* JavaScript event handler, which is triggered when displaying of the page completes.

```
var start = new Date().getTime();
var finish;

function loadtime() {
    finish = new Date().getTime();
    return finish - start;
}

function displayTime() {
    var ok= confirm ("Milliseconds since page started
                    loading at " + start + ": " + loadtime() +
                    ", finished loading at " + finish);
    if (!ok)
        displayTime();
}
```

Figure 4.3 JavaScript Code to Measure and Display Page Rendering Time

With this method, the server processing time to return the Web page (*generation*,  $T_g$ ), the total time taken to serve the page together with its embedded objects (*transmission*,  $T_t$ ), the time for rendering and displaying the page in the Web browser (*rendering*,  $T_d$ ), as well as response time,  $T_r$ , can be determined. As can be seen from Figure 4.2:

- Rendering time,  $T_d = T_{d2} - T_{d1}$
- Response time,  $T_r = T_{d2} - T_{c1}$

While measurements of rendering and total response times are fairly straightforward, measurements of generation and transmission times are slightly complicated due to the following:

- From the client's perspective, time elapsed between a request and the corresponding response, namely serve time ( $T_s$ ), is actually made up of two components: the time for the server to process the request and respond accordingly,  $T_p$ ; and the time the request and response were transmitted over the network,  $T_t$ . The transmission time for a particular requested page or its embedded object is therefore the difference between  $T_s$  and  $T_p$ , for that page or object.
- The server can start responding to the client even though its processing of the client request is not yet completed. This means *generation* and *transmission* may occur at the same time. Subtracting  $T_p$  from  $T_s$  does not precisely reflect  $T_t$ .
- There are times between successive requests when no previous request is being processed nor served. These *inter-request-idle-times*,  $T_{ir}$ , might be used by the client to set up new connection(s) to the server for parallel requests, or the client could wait for the browser to render the page before making subsequent requests. Thus, subtracting  $T_g$  from the time elapsed between the first request sent and the last byte of response received, namely *perceived serve time* ( $T_{sp}$ ), produces a value that reflects the combination of  $T_t$  and  $T_{ir}$ , rather than  $T_t$  itself only.
- An object could be requested while others are being served by the server and while the client is displaying yet other objects. This is because Web browsers, for example,

Internet Explorer, allows up to four simultaneous connections to a single HTTP/1.0 server, and two simultaneous connections to a single HTTP/1.1 server. This means a client browser can request more than one object from the same server at the same time. Moreover, an object can be requested before previously requested objects are received or displayed completely. Therefore, transmission and processing times for different objects could overlap.

Figure 4.4 further facilitates the explanation. It shows communications between the client and the server for downloading a Web page with one embedded object. This is an “ideal” scenario where the client requests and server processes and responses do not overlap.

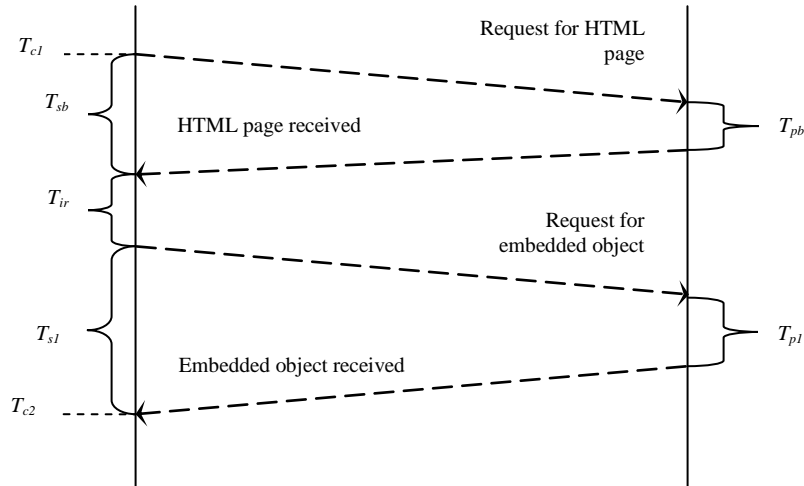


Figure 4.4 Communications between the Client and Server for Downloading a Web Page with One Embedded Object

Assuming transmission time for the HTML page is  $T_{tb}$ , and transmission time for the embedded object is  $T_{tl}$ ; server processing time for the HTML page is  $T_{pb}$ , and server processing time for the embedded object is  $T_{pl}$ . Therefore:

$$T_{sb} = T_{tb} + T_{pb};$$

$$T_{sl} = T_{tl} + T_{pl}; \text{ and}$$

$$T_{sp} = T_{sb} + T_{sl} + T_{ir} = T_{tb} + T_{pb} + T_{tl} + T_{pl} + T_{ir}$$

Generalising the equation for an HTML page with  $n$  embedded objects:

$$T_{sp} = (T_{tb} + T_{pb}) + \sum_{i=1}^n T_{ti} + \sum_{i=1}^n T_{pi} + \sum T_{ir} \quad [\text{Equation 1}]$$

where:

- $T_{ti}$  is the transmission time for the  $i$ -th embedded object.
- $T_{pi}$  is the server processing time for the  $i$ -th embedded object.
- $\sum T_{ir}$  is the total of all *inter-request-idle-time*.

The equation makes most sense when the transmission and processing times for different objects do not overlap. However, when the times do overlap, the equation may produce inaccurate results. This is depicted in Figure 4.5.

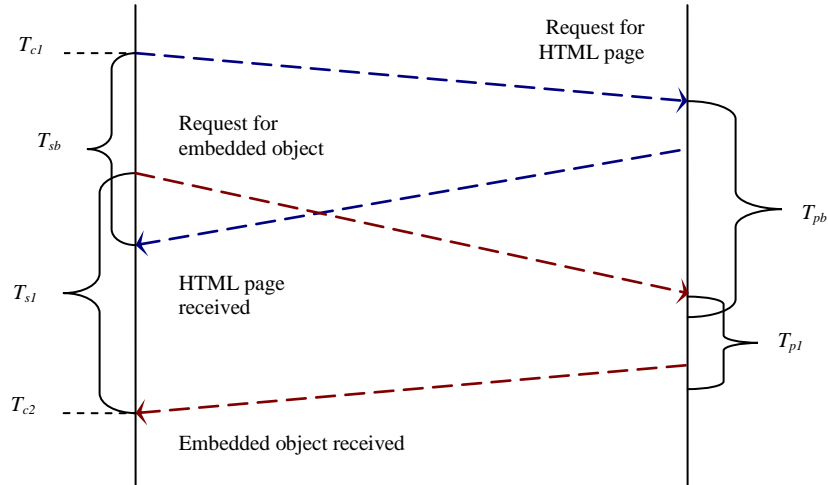


Figure 4.5 Communications between the Client and Server with Overlapping Transmission and Server Processing Times

In this example,  $T_{sb}$  overlaps with  $T_{sl}$ . Therefore, using Equation 1 will produce a value that is larger than the actual  $T_{sp}$ , measured as the difference between  $T_{c2}$  and  $T_{c1}$ , as shown in Figure 4.2. Moreover, when the response started to arrive at the client and the time of that arrival was recorded, the server may yet need to finish processing the corresponding request.

The server access log, on the other hand, will only record the completion time when the processing finishes. In certain cases, it might happen that a  $T_p$  is greater than the corresponding  $T_s$ . Therefore, if  $T_g$  is calculated as the sum of all server processing time, it may produce a  $T_g$  that is larger than  $T_{sp}$  too. This is particularly possible for dynamically generated Web pages where its  $T_p$  is normally large. It is even more possible when the transmission speed is fast enough to make  $T_t$  for requests and responses small. For example, as shown in Figure 4.5,  $T_{pb}$  is larger than  $T_{sb}$ . For either case where the sum of all  $T_s$  or  $T_p$  exceeds  $T_{sp}$ , it contradicts with the definition of  $T_{sp}$  as *perceived* serve time.

The main issue is that there is, as yet, no way to clearly segregate generation, transmission, and rendering as three independent measurable entities. There is no simple solution to this challenge as generation, transmission, and rendering times are not independent of each other. One of the possible solutions might be to fine-grain timestamps in the server access log at millisecond resolution in order to identify the overlapping periods between server processing and transmission. However, this will eventually involve extensive calculation and the clocks between the server and the client would need to be synchronised too. It is infeasible to implement such a method merely for the sake of more accurate response time measurements.

After considering all these problems, the following definitions for generation and transmission times are adopted:

- Generation time,  $T_g = T_{pb} + \sum_{i=1}^n T_{pi}$ , where  $T_{pb}$  is the server processing time for the HTML page (the base object);  $T_{pi}$  is the server processing time for the  $i$ -th embedded object; and  $n$  is the number of embedded objects in the HTML page. This definition

refers to the cumulative processing time for the HTML page and its embedded objects.

- Transmission time,  $T_t = T_{sb} + \sum_{i=1}^n T_{si}$ , where  $T_{sb}$  is the serve time for the HTML page (the base object);  $T_{si}$  is the serve time for the  $i$ -th embedded object; and  $n$  is the number of embedded objects in the HTML page. Here, transmission time adopts the definition of cumulative serve time for the HTML page and its embedded objects.

As explained earlier, these two definitions may possibly produce  $T_g$  and  $T_t$  that are larger than  $T_{sp}$ . In order to deal with this eventuality, three other notions are introduced to provide greater insights into the measured times. These notions give an additional indication of the significance of the transmission and generation phases to  $T_r$ . The notions are:

- Effective serve time,  $T_{se}$ , calculated by subtracting inter-request-idle-time,  $T_{ir}$ , from perceived serve time,  $T_{sp}$ . It has to be noted that  $T_{ir}$  measured using the Muffin proxy server method is an approximation of the inter-request-idle-time because it records the times the responses started to arrive at the client rather than the completion of the responses.
- The ratio of  $T_{se}$  to  $T_{sp}$ , is denoted as  $T_{se}/T_{sp}$ . It indicates the effectiveness of transmission over the network. A small  $T_{se}$  relative to  $T_{sp}$  denotes effective transmission. However, a huge  $T_{se}$  may imply ineffective transmission and/or generation.
- The ratio of transmission time,  $T_t$ , to generation time,  $T_g$ , is denoted as  $T_t/T_g$ . The larger the ratio, the greater the contribution of transmission to the  $T_r$ . A ratio of less than 1 indicates that generation is a greater determinant to  $T_r$ , as compared to transmission.

Appendix 1 lists the notations used in this section.

#### **4.2.1 Muffin HTTP Proxy Server**

The Muffin proxy server is the key component of this method. It is written in Java, with over 21 000 lines of code, 128 Java classes, and 12 Java interfaces. The goals of the Muffin proxy server arise from privacy and security concerns, since it allows the users to filter unwanted elements and features at HTTP level (Boyns, 2000). Such elements and features include cookies, ActiveX objects, GIF animations, Web advertisements, marquees, and so on. It also allows the user to implement new features, such as the response time measurement method used above. A desirable advantage of the Muffin proxy server is that it does not require modification to the browser or server. It acts as a filter between the Web client and server. The filtering is done at the application layer of the TCP/IP protocol suite, with the default TCP port number 51966.

Client requests and server responses will be parsed by the Muffin proxy server before they are delivered to the intended destination. When an HTTP request or response reaches the proxy server, it will be managed by a handler. Muffin creates a handler for each HTTP transaction and invokes filters for requests, responses, and content relevant to the transaction. A filter manager is in charge of managing all the filters, where it maintains the list of supported filters, enabled filters, and filter preferences. Filtering of the requests, responses, and content is performed by the enabled filter(s). The filtering process inevitably introduces overhead to the measurement of response time. Appendix 2 shows how the HTTP requests and responses are parsed by the Muffin proxy server.

Implementation-wise, the Muffin proxy server should be placed as close to the client browser as possible, so that the recorded times will be as close as possible to the actual times the requests were sent and responses were received by the browser. Appendix 3 shows the code for the response time measurement filter.

#### **4.2.2 Advantages and Disadvantages**

The advantages of this method include:

- It is able to measure response time as experienced by the end users.
- Response time for different Web pages can be measured and compared.
- Pages do not need to be accessed through instrumented hyperlinks as compared to the method used by Rajamony and Elnozahy (2001).
- Deployment of the Muffin proxy server is transparent to end-users. Eliminating the JavaScript instrumentation in the Web page that measures rendering time can make the method even more transparent and non-intrusive.
- Time taken to download an HTML page and each of its embedded objects can be measured separately. Therefore, when an existing page is modified, such as adding new elements, or changes made to the network and/or the server, or any other modifications applied to the existing system, their effect on Web page response time can be examined.
- Time taken by the server to process each HTTP request can be determined. Thus, it supports comparison of processing load for different HTTP requests.
- Transmission time can be calculated.
- Different components of response time can be identified. This can be utilised to isolate and analyse reasons for delays.



The method nevertheless has a few disadvantages:

- The Muffin proxy server inevitably introduces overheads. The Muffin proxy server will parse both the outgoing requests and incoming responses, which affects the times measured for transmission and rendering. Because of the overheads, the measurements do not show absolute response times for an HTML page and its embedded objects. On the contrary, the measurements reflect only relative response times. However, absolute response times are not essential for the purpose of response time analysis. Relative response times are sufficient to compare individual impact of the HTML page and its embedded objects to the total response time.
- Since the server access log is required to support the measurement and analysis, clients with their names or IP addresses hidden by a proxy server can't be identified. If this happens, the measurement and analysis will become more difficult and may be less accurate.
- Deployment of the method in the real world might be a great challenge as it involves, for example, the Web site administrator permitting access to the log, ISP or LAN administrator for the Muffin or other proxy server with similar functionality, and the end-users' browsers for supporting JavaScript instrumentation. It can, however, be deployed with active probing (see Section 2.6.1) to provide more details about Web page response time.

### **4.3 The Second Method: Web Page with Frames**

The most accurate response time measurement method is one that measures responses at the client's browser. The second method takes this approach. It is similar to the previous one, but it replaces the Muffin proxy server with a JavaScript-instrumented Web page, which acts as a

Web browser. The Web page has two frames, one of which is invisible to the end-users. Thus, it appears as a page without frames. It is illustrated in Figure 4.6.

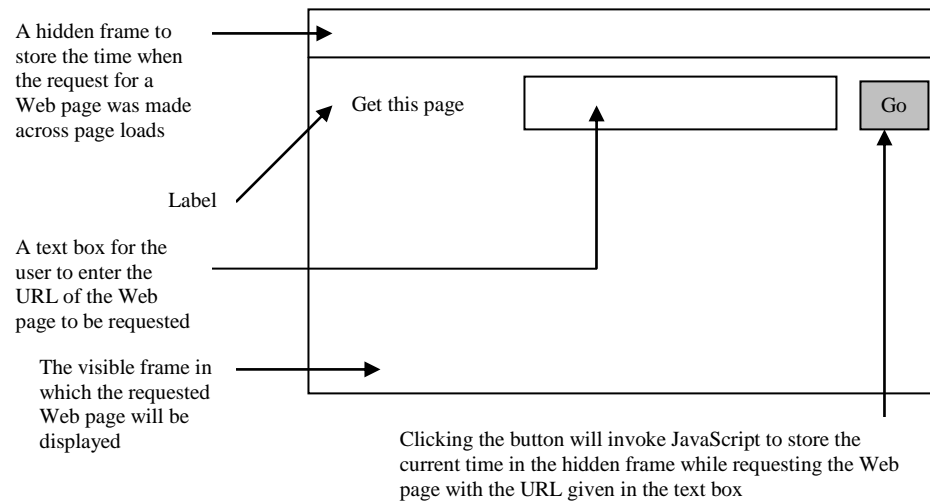


Figure 4.6 Web Page with Two Frames for Measuring Response Time

The reason for using frames is to store data across page loads, which is one of the main challenges measuring response time at the client. When a new page is loaded by the Web browser, whatever data that is stored in the previous page will be cleared away, unless there is a way to pass the values to the new page, or to allow the values to persist. Using cookies<sup>7</sup> is the most common way of passing values from one page to another. Here I propose an alternative approach which fragments a Web page into two frames: a hidden and a visible frame. Requested Web pages are loaded into the visible frame while the hidden frame, together with the values it stores, remains constant across page requests. As compared to using cookies, this simple yet efficient method requires no modification to the requested Web pages, nor to the server that hosts the Web pages, and does not impact response time by requiring a cookie to accompany each request.

<sup>7</sup> Parcels of textual information sent from a Web server to a browser and then returned unchanged by the browser each time it visits that server.

With the page-with-frames response time measurement method, the user first enters the URL of the Web page to be requested into the text box in the visible frame, which mimics the address bar of a normal Web browser. The *Go* button next to the text box acts as the *Go* button usually seen beside address bar of Web browsers like Internet Explorer and Firefox. When the button is clicked, it invokes the JavaScript that sets a variable defined in the page contained in the hidden frame to record the time of that action while sending the request for the page with the URL given in the text box. As the hidden frame still exists when the requested page is loaded into the visible frame and replaces the existing one, the time when the request was made can be stored and brought forward across page loads.

This method also uses JavaScript to store the time instances when the page being loaded into it starts and completes displaying. The initial page loaded into the visible frame (the one with a text box and *Go* button) needs to be unloaded before the requested page is loaded. When it is unloaded, it will trigger the *onUnload* JavaScript event handler. The event handler is used to set another variable defined in the page contained in the hidden frame to indicate the time the page was unloaded. This time is also taken as the time when the requested page starts displaying. When the requested page completes displaying, the *onLoad* event handler will be triggered and records the completion time. Figure 4.7 shows parts of the JavaScript code for this method.

As can be seen from the code, this method measures response and rendering (display) times for the requested page. However, without the Muffin proxy server, it does not measure the time taken to download the requested page and each of its embedded objects as the previous method does. Apart from that, it can provide similar functionality to the previous method, if used together with the server access log.

```

/*This is in the visible frame*/
/*startDisplay and startRequest are defined in the hidden frame*/

var finishDisplay;
var displayTime;
var responseTime;

function setTimes() {
    finishDisplay = new Date().getTime();
    displayTime = finishDisplay -
top.frames['hidden'].startDisplay;
    responseTime = finishDisplay -
top.frames['hidden'].startRequest;
}

function getTimes() {
    var ok= confirm ("Start request at: " +
top.frames['hidden'].startRequest + " ms\n" +
        "Start display at: " +
top.frames['hidden'].startDisplay + " ms\n" +

"Finish display at: " +
        finishDisplay + " ms\n" +

"Display time = " + displayTime + " ms\n" +

"Response time = " + responseTime + " ms\n" .

```

Figure 4.7 JavaScript Code for the Page-with-Frames Method

#### 4.3.1 Advantages and Disadvantages

Below are the advantages of this method:

- The overhead of measurement is low. Only JavaScript is used to record and display the times while requests and responses are handled as usual. Therefore, the measured response time is very close to the actual response time end-users will experience.
- The cost of deployment is minimal. The page with two frames and instrumented with JavaScript requires no additional setup in the client Web browser.
- The requested pages need not be instrumented. This makes it possible to use the method to measure response and rendering times for any Web pages.

There are some disadvantages of this method, on the other hand:

- Every user request for a Web page must be made through the instrumented two-frame page. Otherwise, the times cannot be measured. Unless the functionality is incorporated into Web browsers, it will be difficult to ensure that the end-users use this method to download the Web pages.
- Only response and rendering times for the Web page will be measured. Unlike the Muffin proxy server method, the time elapsed between individual requests that make up the page and its corresponding response cannot be determined.

#### **4.4 Measurement and Decomposition of Response Time**

The two measurement methods were used to measure and decompose Web page response time. For that purpose, a prototype Web site for an online bookstore was to act as the targeting server and accessed from another machine that acts as the client. The details of the setup are as follows:

- The server machine was a Pentium 4-2.40GHz with 512MB main memory, running Apache Tomcat 5.0 on Windows XP, supporting HTTP/1.1 protocol.
- The client machine was a Pentium 3-450MHz with 256MB memory, connected to the server on a 1Gb LAN. The operating system was Windows XP and the Web browser was Internet Explorer 6.0.
- Dynamic Web pages of the online bookstore were developed using Java Server Pages (JSP)<sup>8</sup>.
- Oracle 10g was used for database management.

---

<sup>8</sup> JSP is a Java technology for dynamically generating HTML or other types of documents in response to a Web client request.

Two Web pages were studied using the two methods. The first one was a dynamically generated home page while the second one was a static home page. In terms of content and layout, both pages were identical. As the conditions of the client, server, and LAN were maintained to be as consistent as possible, the times measured for the two pages can be attributed to the way they were created and served – static or dynamic, but not to other factors. Figure 4.8 shows the dynamic version of the home page for the online bookstore Web site.

Different scenarios were examined for each of the home pages. For each scenario, 20 measurements were taken. The scenarios are explained in the next two sub-sections.



Figure 4.8 Online Bookstore Home Page

#### 4.4.1 The Dynamic JSP Home Page

The dynamic home page was developed using JSP. An important feature of JSP is that the first access to a JSP page after the hosting server has started requires compilation of the page into Java source code and then to a Java Servlet. The compilation takes a significant

amount of time and makes the first access slow. Therefore, the first access to the JSP home page was examined as a special case pertinent to the technology.

For other accesses to the JSP home page, three scenarios were studied:

- The first scenario was accesses without caching of embedded objects. The objects are requested and served from the server, and indicated with HTTP status code<sup>9</sup> *200* (OK) in the server access log.
- The second scenario was accesses with caching of embedded objects. The requested objects required validation of their currency from the server before they were retrieved from the cache. Such a request is indicated with an HTTP status code *304* (Not Modified) showing that the requested object is still up-to-date.
- The third scenario was accesses with caching of embedded objects. However, the requested objects did not require validation from the server, meaning that they were retrieved immediately from the local cache when found. These requests will not be recorded in the server access log. With the Muffin proxy server method, neither request nor response will pass through the proxy server. Thus, only the page's rendering time can be measured.

#### **4.4.2 The Static HTML Home Page**

Three scenarios were studied for this static HTML home page.

- The first scenario was accesses without caching of embedded objects. The objects are requested and served from the server, and indicated with HTTP status code *200* (OK) in the server access log.

---

<sup>9</sup> HTTP status code represents a brief description of the response to a client request, such as the request is handled successfully, redirection required, or error occurred.

- The second scenario was accesses with caching of embedded objects. The requested objects required validation of their currency from the server before they were retrieved from the cache. Such a request is indicated with an HTTP status code 304 (Not Modified) should the requested object is still up-to-date.
- The third scenario was similar to the first one except IP address of the server is used instead of its URL for the accesses. The difference between  $T_r$  measured for the first and this scenarios is attributed to DNS lookup where URL is resolved into its corresponding IP address.

## 4.5 Discussion

Times for different access scenarios were measured. For each scenario, 20 measurements were taken and the means are presented here for discussion. Measurements were made using both Muffin proxy server and page-with-frames methods.

### 4.5.1 First Access to the JSP Home Page

Table 4.1 shows the differences between the first access to the JSP home page and the subsequent access to the page without caching of the embedded objects.

Table 4.1 First and Subsequent Accesses to the JSP Home Page in Milliseconds

Time	First Access to the JSP Home Page		Subsequent Access to the JSP Home Page without Caching	
	Muffin	Page-with-Frames	Muffin	Page-with-Frames
$T_r$	4805	4610	541	357
$T_d$	2490	2291	447	307
$T_t$	2324	-	84	-
$T_g$	4472	4428	272	282
$T_{se}$	2299	-	80	-
$T_{sp}$	4663	-	400	-
$T_{se} : T_{sp}$	0.49	-	0.20	-
$T_t : T_g$	0.52	-	0.31	-



For the measurements taken with the Muffin proxy server, the first access to the JSP home page takes 8.9 times (4805:541) longer to complete the download than the subsequent access. Using the page-with-frames method, the ratio is 12.9 (4610:357). This difference is due to the overhead imposed by the Muffin proxy server. The overhead is 195ms for the first access while for the subsequent access, it is 184ms. These are equivalent to 4.2% (195/4610) for the first access and 51.5% (184/357) for the subsequent accesses. The overhead is somewhat low and consistent but since the response time for the subsequent accesses is low (due to the server not being loaded with many client requests concurrently and the transmission taking place on the LAN), the overhead becomes comparatively significant.

The need to compile and interpret the JSP page for the first access caused the longer  $T_r$  measured than the subsequent accesses. Fortunately, this is only experienced by the first access after the server has started. Once the Java servlet has been created,  $T_r$  can be greatly improved.

For the first access, measured with the Muffin proxy server, the ratio of  $T_t:T_g$  is 0.52, means that server processing plays an important role in determining  $T_r$ . Further investigation found that the time taken for the server to respond to the first client request and return the response to the client was the cause of the long delay. Processing for the first client request took 4462ms, which is about 99.8% (4462/4472) of the  $T_g$ . Meanwhile, the first response was received averagely in 2223ms, which is about 95.6% (2223/2324) of the  $T_t$ . There are two implications that can be made from these observations:

- Server processing time for the generation of the dynamic JSP home page was too large as compared to the processing time for its embedded objects. As a result, processing

time for the embedded objects becomes the least significant part of the total response time.

- The client had to wait much longer to receive the first response from the server than other responses. This was also due to the long processing time for the first request, i.e. generation of the JSP home page.

On the other hand,  $T_{se}$  (2299ms) was 25ms less than  $T_t$  (2324ms). It shows very little overlapping between different pairs of request and response and very little  $T_{ir}$ .  $T_{se}:T_{sp}$  was 0.49, shows moderately effective transmission, but again, fairly affected by the delay in server processing.

For subsequent access, it is quite different in terms of  $T_t:T_g$ , which was 0.31 compared to 0.52 for the first access. Processing for the first client request took 267ms, which is about 98.2% (267/272) of the  $T_g$ . The first response was received averagely in 15ms, which is about 17.9% (15/84) of the  $T_t$ . These exemplify two conclusions:

- Server processing for the dynamic JSP home page is still the major contributor to the  $T_g$ .
- However, the client could receive the first server response within relatively shorter time since the servlet had been created and thus the server could respond to the client more quickly. This is further evidenced by the  $T_{se}:T_{sp}$ , which was 0.20, demonstrating very effective transmission.

In this issue pertinent to Java technology, it has been shown from the measurements that the importance of pre-compiling Java source code into executable objects whenever possible, and

the impact of such advanced compilation on Web page response time, especially improvement on generation of dynamic JSP pages.

#### 4.5.2 Caching

The effect of caching on response time was studied for both dynamic and static pages. Subsequent accesses to the dynamic page with and without caching were used to examine the effects of caching. The result is shown in Table 4.2.

Table 4.2 Subsequent Accesses to the JSP Home Page with and without Caching in Milliseconds

Time	Subsequent Access to the JSP Home Page without Caching		Subsequent Access to the JSP Home Page with Caching	
	Muffin	Page-with-Frames	Muffin	Page-with-Frames
$T_r$	541	357	464	288
$T_d$	447	307	376	235
$T_t$	84	-	58	-
$T_g$	272	282	214	203
$T_{se}$	80	-	58	-
$T_{sp}$	400	-	324	-
$T_{se} : T_{sp}$	0.20	-	0.18	-
$T_t : T_g$	0.31	-	0.27	-

Comparing the times measured using the two different methods, the overheads imposed by the Muffin proxy server are 184ms for access without caching and 176ms for access with caching. The overhead is still considered high compared to the low response time measured in the LAN environment. Using the times measured with page-with-frames method, it is found that caching improves response time by 69ms, or 19.3% (69/357). Other components, including  $T_g$ ,  $T_t$ , and  $T_d$  are improved too. However,  $T_{se} : T_{sp}$  and  $T_t : T_g$  improve slightly only, with improvement on  $T_t$  (31.0%) exceeds improvement on  $T_g$  (21.3%). This is explained because caching mainly reduces transmission of the objects from the server to the client where cached objects can be retrieved locally at the client. Meanwhile, it also relieves the

server from tasks such as finding the requested files or objects and writing responses to the output buffer, and hence reduces  $T_g$ .

Caching policy is an important factor that affects the effectiveness of caching. Cached objects were validated by the server before the client browser fetched and displayed them. This validating process introduces communication load between the client and the server, which eventually adds to the  $T_r$ . If the caching policy is changed so that cached objects are used without validating from the server,  $T_r$ , as measured by the page-with-frames method, was reduced to 98ms. This is 190ms (66.0%) less than the  $T_r$  for caching with validation policy. Table 4.3 gives a picture of the effect of caching on the static page.

Table 4.3 Accesses to the HTML Home Page with and without Caching in Milliseconds

Time	Access to the HTML Home Page without Caching		Access to the HTML Home Page with Caching	
	Muffin	Page-with-Frames	Muffin	Page-with-Frames
$T_r$	328	172	291	147
$T_d$	208	119	188	91
$T_t$	78	-	67	-
$T_g$	5	5	5	6
$T_{se}$	74	-	64	-
$T_{sp}$	263	-	220	-
$T_{se} : T_{sp}$	0.28	-	0.29	-
$T_t : T_g$	15.6	-	13.4	-

The overheads imposed by the Muffin proxy server are 156ms for access without caching and 144ms for access with caching. The magnitude of the overhead is a little less than the overhead for the accesses to the dynamic page, showing the difference particularly in the ways dynamic and static pages are served by the server to the client. As dynamic pages are generated on the fly by the server and transmitted to the client at the same time, they could result in more data packets to be transmitted compared to their identical (in terms of content) static counterparts, given the fact that the sizes of IP packets are variable due to the packet

payloads with different lengths. The more data packets transmitted to the client means the more parsing by the Muffin proxy server involved and that increases the overhead.

With the support of caching,  $T_r$  is improved by 25ms, or 14.5% (25/172). The improvement is smaller than for the accesses to the dynamic page. For accesses to the dynamic page, the improvement that caching has can be seen on both  $T_g$  and  $T_t$ . However, for accesses to the static page, the room for improvement on  $T_g$  is too small and caching only improves  $T_t$ . This is demonstrated by the reduction of  $T_t:T_g$  from 15.6 to 13.4 while there is no improvement on  $T_g$ .

#### 4.5.3 Dynamicity of Web Page

There is no doubt that the JSP home page will have longer download delay than the static HTML home page. However, to what extent do the two pages differ from each other in terms of transmission, generation, and rendering times? Table 4.4 shows the times measured for accesses to the dynamic JSP home page and the static HTML home page.

Table 4.4 Accesses to the JSP and HTML Home Pages without Caching in Milliseconds

Time	Access to the JSP Home Page without Caching		Access to the HTML Home Page without Caching	
	Muffin	Page-with-Frames	Muffin	Page-with-Frames
$T_r$	541	357	328	172
$T_d$	447	307	208	119
$T_t$	84	-	78	-
$T_g$	272	282	5	5
$T_{se}$	80	-	74	-
$T_{sp}$	400	-	263	-
$T_{se} : T_{sp}$	0.20	-	0.28	-
$T_t : T_g$	0.31	-	15.6	-

Referring to the times measured using the page-with-frames method where the measurement overhead is smaller than using the Muffin proxy server method,  $T_r$  is reduced by 185ms, or

51.8% (185/357). It can be seen that  $T_g$  is reduced to the largest extent (98.2%). This is not surprising at all as generating a dynamic page is a much more intensive task than serving a static page. Since  $T_g$  is reduced greatly,  $T_t$  has become more significant in contributing to  $T_{sp}$ , as depicted by the higher  $T_t:T_g$ , from 0.31 to 15.6. Even though  $T_t:T_g$  is much larger than 1, moderate  $T_{se}:T_{sp}$  at 0.28 shows transmission was still effective as it took place in the LAN environment where bandwidth and transmission efficiency is seldom a factor that deteriorates  $T_r$ .

#### 4.5.4 DNS Lookup

DNS is the process that resolves a URL into its corresponding IP address. DNS lookup time is part of the  $T_r$  measured but it cannot be measured directly as a single and independent component by either the Muffin proxy server method or the page-with-frames method. Nevertheless, indirect inference of DNS lookup time can be made if the IP address for a particular URL is known. Table 4.5 compares the times measured for the accesses to the static home page without caching, by using URL and IP address respectively.

Table 4.5 Accesses to the HTML Home Page without Caching Using URL and IP Address in Milliseconds

Time	Access to the HTML Home Page without Caching (URL)		Access to the HTML Home Page without Caching (IP Address)	
	Muffin	Page-with-Frames	Muffin	Page-with-Frames
$T_r$	328	172	297	153
$T_d$	208	119	197	106
$T_t$	78	-	75	-
$T_g$	5	5	5	5
$T_{se}$	74	-	69	-
$T_{sp}$	263	-	234	-
$T_{se}:T_{sp}$	0.28	-	0.29	-
$T_t:T_g$	15.6	-	15.0	-

According to the measurements made using the page-with-frames method,  $T_r$  is reduced by 19ms, or 11.0% when an IP address instead of the URL is used to access the static home page.

This indicates the DNS lookup time. The measurements made using the Muffin proxy server method is then used to examine different components of response time. It is found that  $T_{sp}$  is improved the most (29ms or 11%) compared to  $T_d$  (11ms or 5.3%),  $T_t$  (3ms or 3.8%),  $T_g$  (-), and  $T_{se}$  (5ms or 6.8%). This means responses were served within shorter period after the request was made, due to the time saved on DNS lookup.

#### 4.5.5 Rendering and Correlations between Different Time Components

While generation time is mainly affected by the server's load and capability, and transmission time is basically a result of network speed and capacity, rendering time is not only determined by the client browser. Apart from the content and structure of the page being rendered, it also depends on how fast the page and its embedded objects can be served, which in turn is influenced by the efficiency of generation and transmission. This brings forward the problem that rendering time can easily be measured but there is a need to isolate the significance of the rendering task from transmission and generation in contributing to the time measured.

One of the ways to deal with this problem is to study the correlation between  $T_d$ ,  $T_t$ , and  $T_g$  to see how they are related to each other. Table 4.6 shows the correlation coefficients between them for different access scenarios.

Table 4.6 Correlation Coefficients between  $T_t$ ,  $T_d$ , and  $T_g$

Access Scenario	$T_t - T_d$	$T_t - T_g$	$T_d - T_g$
First Access to JSP Home Page	-0.11	0.19	0.72
Subsequent Access to JSP Home Page without Caching	-0.03	0.24	0.75
Subsequent Access to JSP Home Page with Caching	-0.15	0.11	0.67
Access to HTML Home Page without Caching	0.08	0.13	0.43
Access to HTML Home Page with Caching	0.22	0.18	-0.23

A few observations can be drawn from the data:

- $T_t$  does not closely relate to  $T_d$  or  $T_g$ . The low or even negative relationship between  $T_t$  and  $T_d$  or  $T_g$  is a result of the way  $T_t$  is defined and measured.  $T_t$  is in effect the cumulative time to serve individual Web page and its embedded objects. Meanwhile, the serve time measured only counts from the moment the request was sent until the corresponding response started to arrive, rather than the complete recipient of the response.
- If  $T_{sp}$  is used instead of  $T_t$  to indicate transmission, with the assumption that  $T_{ir}$  is small and ignorable, the correlation between transmission and other time components will be different, as can be seen in Table 4.7. Correlations between  $T_{sp}$ ,  $T_d$ , and  $T_g$  are mostly positive and more essential as compared to the correlations between  $T_t$ ,  $T_d$ , and  $T_g$ , except for the access to the static HTML page with caching.  $T_{sp}$  and  $T_g$  exhibits the closest correlation for the accesses to the JSP home page.
- For accesses to the HTML home page,  $T_d$  is more closely related to  $T_{sp}$  than  $T_g$ , and vice versa for accesses to the JSP home page.

Table 4.7 Correlation Coefficients between  $T_{sp}$  and  $T_r$ ,  $T_d$ , and  $T_g$

Access Scenario	$T_{sp} - T_d$	$T_{sp} - T_g$
First Access to JSP Home Page	0.57	0.48
Subsequent Access to JSP Home Page without Caching	0.66	0.92
Subsequent Access to JSP Home Page with Caching	0.49	0.70
Access to HTML Home Page without Caching	0.54	0.27
Access to HTML Home Page with Caching	0.76	-0.38

Such correlation coefficients can be used to examine performance for a Web based system. For example, if the correlation of  $T_d$  for a Web page with its  $T_g$  is higher than the correlation in ideal condition, it is most probably caused by poor server performance in serving the page.



## 4.6 Conclusion

Two Web page response time measurement methods have been introduced in this chapter. The two methods serve the same purpose with different strengths. The Muffin proxy server method facilitates detailed scrutiny of the measured response time. Response times for an HTML page and its embedded objects can be measured separately. Therefore, the impact of the HTML page itself and each of its embedded objects to the total response time can be identified and compared. In spite of the measurement overheads introduced by the tool, the measurement results are useful in evaluating relative response time for the HTML page and its embedded objects. In addition, deployment of the measurement tool can be made independently of the Web server and client. Meanwhile, the page-with-frames method has low measurement overhead and accurately measures total response time experienced by the end-users. Even though it does not show the individual impact of HTML content and embedded objects to the total page response time, it is a simple yet efficient measurement tool that requires no modification to either the Web server or the requested Web pages. It has also been demonstrated that the two methods can be used to examine different issues pertinent to Web page response time, such as caching, DNS lookup, and dynamicity of Web pages. The next chapter focuses on modelling of Web page response time. Web page characteristics that influence response time are grouped into two categories of complexity: *creation* and *content*. Each category is further divided into three complexity dimensions. Response time is expressed as models in terms of these complexity dimensions. The page-with-frame measurement method presented in this chapter is then used to verify the models.

## CHAPTER 5

### MODELLING WEB PAGE RESPONSE TIME

This chapter explains the process to model Web page response time as a function of its complexity. Here, the term *complexity* refers to Web page characteristics that reflect two main issues related to the Web page: the way it is created and its content. Section 5.1 explains the concept of modelling. Section 5.2 examines the relationship between Web page complexity and response time. Web page complexity is divided into two categories, i.e. *creation* and *content*, and discussed in Sections 5.3 and 5.4, respectively. Section 5.5 describes how models of response time in relation to Web page complexity are built and validated. Section 5.6 concludes the chapter.

#### 5.1 Modelling

A model is a representation of the real world (Maani & Cavana, 2003). In computing, modelling is the technique often used to give simplified descriptions of computer systems (Crovello & Krishnamurthy, 2006). However, as claimed by Floyd and Kohler (2002), “*models should be specific to the research questions being investigated*” (p. 1), especially when modelling aspects of the Internet. Statistician George Box (Box, 1979) even asserts that “*all models are wrong but some are useful*” (p. 202) because there is no model that could exactly represent any system in the real world. In general, the two points below can be made about a model:

- A model is an approximate rather than exact representation of a system.
- A model only includes parameters that are of its intended interest and the rest might be omitted.

The two points refer to two important requirements of a model: *simplification* of the real world system, and *relevance* to the field of interest. In order to fulfil these requirements, parsimonious models, which contain small number of parameters, are preferable. An advantage brought forward by parsimony is to reduce the possibility that improvement in fit of data to the model is caused by additional degrees of freedom with the inclusion of more parameters, rather than the model itself is a better choice to fit the data (Crovella & Krishnamurthy, 2006). Furthermore, models with fewer parameters are not only easier to interpret, but also tend to increase precision of the model (Box, 1979).

Modelling in the context of computing is essentially comparable to the third level of the thinking hierarchy, i.e. systematic structure. Thinking hierarchy is illustrated as a four-level pyramid as shown in Figure 5.1 (Maani & Cavana, 2003).

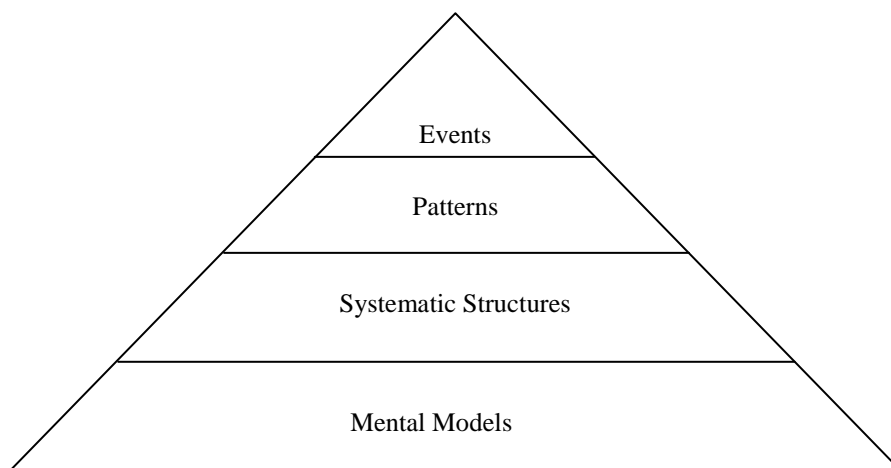


Figure 5.1 Four Levels of Thinking Hierarchy

A systematic structure summarises the pattern exhibited by informative but shallow events. This is similar to a model that describes the pattern drawn by a set of measured data. Meanwhile, a mental model is akin to underlying assumptions that govern the model.

If a model is derived from a set of measured data that summarises the dataset by some statistical parameters, it is called a descriptive model. If a model is a description of a process or system elements that could have ideally produced the dataset to be examined, then it is called a constructive model (Salamatian & Fdida, 2003). A descriptive model views the system as a black box and a constructive model takes a white box view to the system. In either case, a model is a useful complement to measurements as well as simulation. For example, a descriptive model can be used to interpret data obtained from measurements while a constructive model can be used to produce input data for simulations. As discussed in Chapter 2, measurement, simulation, and modelling are the three fundamental performance analysis techniques.

## **5.2 Web Page Complexity and Response Time**

Complexity is a term that is normally used to refer to presentation of a Web page. For example, Bucy *et al.* (1999) associate complexity with the design and features of a Web page, including graphical and dynamic elements, real-time and asynchronous interactivity, frames, screens, page maps, visitor counters, advertisements, banners, hyperlinks and background colours. Geissler *et al.* (2001) on the other hand, use complexity to refer to the number of links, number of graphics, home page length, and animation. Germonprez and Zigurs (2003) extend the scope of complexity beyond structural dimensions of a Web site and define complexity as a result of human cognition required to use the site, content (particularly information) available at the site, and the form in which the site is constructed.

These studies, directly or indirectly, relate complexity of a Web page or Web site to its usability<sup>10</sup>. In fact, usability is one of the fields that is most often associated with the complexity or design of a Web page. Among others:

- Chadwick-Dias *et al.* (2003) study the effect of Web page design on the users' performance in carrying some given tasks, especially users 55 years of age or older.
- Ward & Marsden (2003) examine users' physiological measurements, including skin conductance, heart rate, and finger blood volume when presented with different human-computer interaction (HCI) events. The HCI events are based on two different versions of a Web site, one well-designed and another ill-designed, against good Web and information design principles.

Another field that Web page complexity is associated with is information retrieval. For instance:

- Larson & Czerwinski (1998) study the optimal levels of depth and breadth in Web site structure that aids the users' information retrieval tasks.
- Song *et al.* (2004) partition a Web page into semantic blocks with a hierarchy structure and identify the importance of each block that could assist in the placement of information in the page according to its significance. Placing significant information in a more prominent block will help the users to locate them more quickly.

Web page complexity is also studied in relation to e-Commerce or marketing:

---

<sup>10</sup> Defined in ISO 9241-11 as “*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*”.

- Mandel & Johnson (2002) investigate the effect of priming, i.e. exposure to some prior event to increase the accessibility of existing information in the subject's memory, on the users' choice when buying online. Priming was manipulated through background pictures and colours of a Web page.
- Debaraj *et al.* (2003) consider Web page design features as one of the factors affecting customer loyalty for online retailing.

Different from the studies mentioned above, this study, however, finds the relationship between Web page complexity and its response time and models of this relationship are formulated. This relationship can be demonstrated with a *causal link* (Maani & Cavana, 2003) as in Figure 5.2.



Figure 5.2 Causal Link between Web Page Complexity and Response Time

A causal link consists of two variables and an arrow. A variable in a causal link can influence, or be influenced by, another variable. The relationship is indicated by an arrow, or link, which represents causal association between the two variables. There are two possibilities in which the two variables are related:

- They move in the same direction, indicated by an 's' at the end of the arrow. Moving in the same direction means increase or decrease in one variable will cause a corresponding increase or decrease in another variable.

- They move in the opposite direction, indicated by an ‘o’ at the end of the arrow. This means increase or decrease in one variable will cause a corresponding decrease or increase in another variable.

Anecdotal statements made about the relationship between Web page complexity and its response time can be found in the literature. For example, Nielsen (1997) suggests reducing page size by having minimal graphics and multimedia effects in order to give prevalence to download speed. Hitz *et al.* (2006) recommend limiting the number of images and other high-volume media elements in a single page so that the total size of the page does not exceed approximately 50 kilobytes (KB). The value is based on a rough calculation<sup>11</sup> of using 56 kilobits per second (Kb/s) Internet connections via modem to download the page and still fulfils the 8-second rule proposed by Zona Research (see Section 2.4). These assertions are in accordance with the causal link shown in Figure 5.2 where increase in the page complexity will result in the increase of response time.

Nevertheless, there is, as yet, no systematic scientific research that studies Web page complexity in association with its response time. This research will fill the gap by studying Web page complexity from two perspectives, i.e. creation and content, and will build models that represent the relationship between Web page complexity and resulting response time.

Before continuing with further discussion, it is worthwhile to give a definition for the term *complexity* that is used in this study. We define complexity as *the means by which a Web page is created (creation) and what it consists of (content) that has a causal effect on its*

---

<sup>11</sup> 8 seconds x (56 Kb/8) = 56 KB

*response time*. Effects of technologies, such as Web server hardware and network bandwidth, on Web page response time are not included as main parameters in the models, even though they are captured in the models when necessary. It is outwith the focus of this research to identify the relative importance of the different complexity dimensions in influencing Web page response time.

### 5.3 Creation Complexity

Creation complexity of a Web page is viewed from two major aspects. First of all, a Web page requested by the client can be created in one of the two ways:

- It is created prior to the request. This is called a *static* page.
- It is created on the fly upon receipt of the request. This is called a *dynamic* page.

These two situations represent two ends of the spectrum of Web page creation methods. The most common pages are those which are a mixture of static and dynamic elements. Some static pages contain a few dynamic components while some dynamic pages are created based on customisation of static templates. Furthermore, dynamic components can be pre-generated before they are actually requested by the end users in order to improve response time. In general, under the same conditions, such as server capability and load, network bandwidth and traffic conditions, serving a dynamic Web page will always take longer than serving its static counterpart; even though their delivered contents may well be identical (see Section 4.5.3).

Secondly, particular to dynamic Web pages, creation of a page may involve database queries. Interaction between the Web server and database will contribute to the Web page response time.



### 5.3.1 Static and Dynamic Pages

For static Web pages, creation complexity can reasonably be considered to be proportionate to lines of (non-comment) code (LOC). It is defined in such a way because static Web pages only involve sequential execution of HTML codes by the browser to layout the pages, and probably some scripts (categorised as content complexity and discussed in Section 5.4). The more lines of code there are, the more bits there will be to transmit from the server to the client, and a longer page to be rendered by the client's browser. Consequently, response time will increase proportionally.

For dynamic pages, other measures need to be considered apart from LOC. This is because dynamic pages may involve programming constructs such as functions and iterations. In this sense, a dynamic Web page is, at least partially, similar to a traditional software program. Different complexity metrics have been proposed to measure different aspects of such software programs, including cyclomatic complexity<sup>12</sup> (McCabe, 1976), program length<sup>13</sup> (Halstead, 1977), and program complexity by the pair<sup>14</sup> (Hansen, 1978). However, the term *complexity* used in those contexts is different from the definition of complexity used in this study. In those contexts, complexity is more a measure of effort to comprehend, test, and maintain software (Carver, 1988) than a metric that is related to response time of the software.

---

<sup>12</sup> A metric that measures logical complexity of a program through examination of the number of linearly independent paths of the program. If the program is represented by a flow graph with  $E$  edges and  $N$  nodes, its cyclomatic complexity is defined as  $V(G) = E - N + 2$ .

<sup>13</sup> A metric that measures a program's expression complexity based on numbers of operands and operators in the program. For a program with  $N1$  operators and  $N2$  operands, the program length is defined as  $N = N1 + N2$ .

<sup>14</sup> A metric to represent complexity of a program with lexicographically ordered 2-tuples consisting of cyclomatic complexity and expression complexity characterised by number of operators in the program.

In this study, a two-level measure is used to express the complexity of a dynamic Web page. The first level is LOC to create the page. It refers to non-comment and executable code only but excludes the code examined in the second level measure, which involves program functions, iterations, and decisions. The code to create a page may be scattered in a few files. If this is the case, the code will be placed together to get the count for LOC. The second level measure for creation complexity involves the Big O notation introduced by German mathematician Paul Bachmann (1837-1920). This is a common notation to deal with approximation of a quantity (Knuth, 1968) and expressed as  $O(f(n))$ . According to Knuth, the notation is used whenever  $f(n)$  is a function of the positive integer  $n$  and every appearance of  $O(f(n))$  precisely means:

*There is a positive constant  $M$  such that the number  $x_n$  represented by  $O(f(n))$  satisfies the condition  $|x_n| \leq M|f(n)|$ . (p. 104)*

Big O notation is used to describe the asymptotic behaviour<sup>15</sup> of functions and allows comparison between different functions. It provides estimation for number of steps, and thus related to time required to solve a problem using a particular algorithm. As a measure for Web page creation complexity, Big O notation is used to approximate and compare the time required to create different dynamic Web pages.

Even though LOC is used as a creation complexity measure for both static and dynamic pages, different weight should be assigned to the code to create a static or dynamic page. A line of code for a static page is normally an instruction to the client browser pertinent to page content and layout. However, a line of code for a dynamic page may involve server processing, which is more time consuming than one for a static page.

---

<sup>15</sup> A method for describing limiting behaviour in algorithm analysis.

Assuming the weight assigned to a line of code for a static page is called the *code weight*, denoted as  $w_l$ , a static page with  $m$  LOC will have the creation complexity of  $mw_l$ . On the other hand, for a dynamic page with  $m$  LOC, the complexity in relation to LOC is  $xmw_l$ , where  $x \geq 1$ . It has to be noted that  $x$  is a value that is dependent on the server and technologies used to create the dynamic page.

The second level creation complexity measure for a dynamic page, which involves the Big O notation, is used as below:

- Decision, iteration, and function are the three programming constructs to be estimated using Big O notation.
- For each construct, the complexity is expressed with Big O notation in the form of  $O(f(n))$ .
- The LOC in the construct is denoted as  $l_p$ .
- Complexity of the construct is expressed as  $f(n) \times l_p \times w_2$ , where  $w_2$  is the execution weight assigned to a line of code of the construct.
- Complexity of all the constructs is represented as  $\sum_{i=1}^j (f(n_i) \times l_{p,i} \times w_{2,i})$ , where  $j$  is the number of constructs for that dynamic page.

There are occasions when certain parts of code in a dynamic page are not available to estimate their complexity. One of the examples would be Enterprise Java Beans (EJB) used to provide certain services to generate the page (see Section 2.5.9). If this happens, an estimated complexity,  $c_e$ , is assigned to each part of that kind to compensate for unavailability of  $f(n) \times l_p \times w_2$ . Determination of the value to be assigned to  $c_e$  requires

specific skills and is somewhat subjective. Thus the use of  $c_e$  should be avoided whenever possible.

With the two-level measure explained above, creation complexity of a Web page, regardless of whether it is static, dynamic, or a hybrid of static and dynamic, can be well represented.

### 5.3.2 State

Information about state is fundamental, yet crucial, in the creation of dynamic Web pages. State is maintained in two different ways – using a database at the server end and/or cookies on the client machine (or other ways for handling session variables, such as URL-rewriting). In relation to database, the following aspects are considered:

- The size of database,  $s_d$ . The larger the database that is accessed, the longer the time it will take to construct the output from a given query.
- Number of connections established,  $t$ . Establishing a connection with the database is a both resource and time consuming task. The more connections established in order to create a Web page, the longer it will take to complete the creation.
- Number of database queries,  $q$ . Instead of merely counting the number of query statements, this measure refers to the number of distinct queries per database table. Thus, if a query statement involves accesses to two database tables, such as that in a *SQL join* statement, the number of queries counted will be two rather than one. Meanwhile, if there are two different queries access the same table, the count will be two as well.

On the other hand, handling session variables and cookies also affects response time. A session specifies the period of time that a user with a unique IP address spends on a Web site. During a session, different variables can be used to maintain information about the user. This is to compensate the fact that being a stateless protocol, HTTP is unable to keep track of the user across multiple connections established between the client and the server within the same session. With those variables, the illusion of a session can be created across a series of sequential page accesses from the same site.

One of the ways to store the session variables is by using cookies. Cookies are small bits of textual information sent from a Web server to a browser and returned unchanged from the browser when it visits the same Web site or domain later (Hall, 2000). Exchanging cookies between the client and server and retrieving session variables or other information from the cookies will increase response time. The effect of session variables and cookies on creation complexity is indicated by  $k$ .

The time to establish  $t$  database connections is independent from the size of the databases, while the size of a database,  $s_d$ , will affect the time required to construct output from a query  $q$  to the database. Therefore, the complexity to reflect database access as well as session variables and cookies takes the form of  $f(t) + g(q, s_d) + k$ . Database performance affected by other factors such as speed of physical disk, database page size, and organisation of the database on physical disks, are not captured in the complexity model.

### 5.3.3 Modelling Creation Complexity

Assuming creation complexity is denoted as  $C_I$ . For static Web pages,  $C_I$  is expressed as a function of LOC. Thus,  $C_I = f(m)$  where  $m$  is the lines of non-comment and executable

HTML code for the page. In general,  $f(m) = mw_l$ , where  $w_l$  is an execution weight assigned to each line of code to reflect the line's execution time.

For dynamic pages,  $C_l$  is expressed as a function of LOC ( $l$ ), programming constructs ( $p$ ), and database access ( $d$ ). Thus,  $C_l = f(l, p, d)$ . The function is further elaborated below:

- A page with  $m$  LOC (non-comment and executable code but excluding code examined in terms of  $p$ ), its creation complexity related to  $l$ ,  $g(l)$ , is defined as  $g(l) = xmw_l$ , where  $x \geq 1$ .
- A page with  $j$  programming constructs (decision, iteration, and function), each has  $l_{p,i}$  LOC and the complexity represented by Big O notation as  $O(f(n_i))^{16}$ , where  $1 \leq i \leq j$ , its creation complexity related to  $p$ ,  $g(p)$ , is defined as  $g(p) = \sum_{i=1}^j (f(n_i) \times l_{p,i} \times w_{2,i})$ , where  $w_2$  is the execution weight assigned to a line of code of the constructs, and  $f(n_i) \times l_{p,i}$  is an approximation for the total LOC to be executed for the construct.
- A page that establishes  $t$  connections to  $j$  databases, each database  $d_i$  with size  $s_{d,i}$ , where  $1 \leq i \leq j$ , and there are  $n_i$  queries for each  $d_i$ , its creation complexity related to  $d$ ,  $g(d)$ , is defined as  $g(d) = w_3t + \sum_{i=1}^j (n_i \times w_{s_{d,i}}) + k$ , where  $w_3$  is the weight that reflects time required to establish a database connection, while  $w_{s_{d,i}}$  reflects the time to process a query made to a database with size  $s_{d,i}$ .  $k$  on the other hand is an indication for the effect of session variables and cookies on response time.

---

<sup>16</sup> Definition of Big O Notation is given in Section 5.3.1.

In this way, the three dimensions related to the creation complexity for a dynamic page, i.e.  $l$ ,  $p$ , and  $d$ , can be quantified. Furthermore,  $C_l$  can be expressed in the form of a 3-tuple, i.e.  $(a, b, c)$ , where  $a$ ,  $b$ , and  $c$  are the quantified values for  $g(l)$ ,  $g(p)$ , and  $g(d)$  respectively.

## 5.4 Content Complexity

In early days of the Web, most pages were purely comprised text and their size was small. Later, graphical images became more popular and widely used, then followed by other multimedia elements such as audio and video. Thus, the size of Web pages tends to be much larger nowadays. This raises two issues about the content of a Web page that has impact on its response time, i.e. the total size of the Web page and the number of embedded objects in the page. Apart from that, elements included in Web pages such as scripts, cascading style sheets<sup>17</sup> (CSS), and Ajax<sup>18</sup> (Asynchronous JavaScript and XML) also affect Web page response time.

Based on these three dimensions, a content complexity,  $C_2$  for Web pages is defined.

- The first dimension is number of embedded objects in the page,  $n_o$ . A page with more embedded objects is classified as a more complex page than one with less embedded objects.  $C_2$  can be defined as a function of  $n_o$ , i.e.  $C_2 = f(n_o)$ .
- The second dimension is total size of the page,  $s_p$ . The larger a page is in terms of its size (measured in bytes), the higher level of complexity it is classified.  $C_2$  can be defined as a function of  $s_p$ , i.e.  $C_2 = f(s_p)$ .

---

<sup>17</sup> A language used to describe the presentation of a structured document, normally a Web page written in a markup language.

<sup>18</sup> A Web development technique for creating interactive Web applications by making Web pages seem to be more responsive through exchanging only small amounts of data, rather than reloading the entire page, with the server behind the scenes upon user requests.

- The third dimension is elements that affect response time,  $e_r$ . Every such element will increase the level of complexity of a page. Each element is assigned a weight to reflect its complexity. One such element is the Cascading Style Sheet (CSS). The complexity weight to be assigned to a CSS can be based on its size in kilobytes or number of rules and declarations it defines. Another example is JavaScript and the complexity weight to be assigned can be based on its LOC or number of operations it performs.  $C_2$  is defined as the sum of complexity weights for all the elements, i.e.

$$C_2 = \sum_{i=1}^j w_{e_{r,i}}, \text{ where } j \text{ is the number of elements involved in the generation of the}$$

Web page and  $w_{e_{r,i}}$  is the complexity weight assigned to element  $e_{r,i}$ . Thus,  $C_2$  can be defined as a function of  $e_r$ , i.e.  $C_2 = f(e_r)$ .

In other words,  $C_2$  is proportionate to all the three dimensions. The function,  $C_2 = f(n_o, s_p, e_r)$  can also be expressed in the form of a 3-tuple, i.e.  $(a, b, c)$ , where  $a$ ,  $b$ , and  $c$  are the quantified values for  $f(n_o)$ ,  $f(s_p)$ , and  $f(e_r)$  respectively.

## 5.5 Applying and Validating the Models

With creation complexity,  $C_1$ , and content complexity,  $C_2$ , defined for a Web page, the page's response time,  $R$ , can be stated as a function of  $C_1$  and  $C_2$ . Thus, a constructive model can be formed:

$$R = f(C_1, C_2)$$

Seven sets of Web pages were designed and the models applied to represent their complexities. Their response times were measured using the page-with-frames method (see Section 4.3) to check against the models. The Web browser used was Internet Explorer 6.0.



Web pages were served from the same server and accessed within the same LAN environment with the server to minimise the effect of external factors, especially network conditions, on response time.

### 5.5.1 Lines of Code

The first set of four static Web pages differ in terms of LOC. Page 1a contains 108 LOC and its size is 2 KB while Page 1b contains 18 LOC and the size is 2kB too. Page 1c has 108 LOC and Page 1d has 1008 LOC. The sizes of these two pages are both 17kB. Each line of code results in a line of text to be displayed on the Web browser. However, a line of code in Pages 1b and 1c results in a longer line of text to be displayed on the Web browser than a line of code in Pages 1a and 1d. Nevertheless, Pages 1a and 1b display the same total number of characters eventually. Pages 1c and 1d also display the same total number of characters in the browser, but more than Pages 1a and 1b do. Page 1d results in the longest page, followed by Pages 1a and 1c, and Page 1b is the shortest page among them. There is no embedded object or elements such as CSS in the pages. Table 5.1 shows details of the pages that are used to model their complexities.

Table 5.1 Details for Pages Differ in LOC

	<b>Page 1a</b>	<b>Page 1b</b>	<b>Page 1c</b>	<b>Page 1d</b>
LOC	108	18	108	1008
Programming Constructs	0	0	0	0
State	0	0	0	0
No. of Embedded Objects	0	0	0	0
Total Page Size (kB)	2	2	17	17
Elements	0	0	0	0

Since the four pages are static, their  $C_l$  is a function of LOC in the form of  $mw_l$  where  $m$  indicates LOC and  $w_l$  indicates execution weight assigned to a line of code. Thus,  $C_l$  for Pages 1a, 1b, 1c, and 1d are  $108w_l$ ,  $18w_l$ ,  $108w_l$ , and  $1008w_l$  respectively. 10

measurements of response and rendering times were taken for each page. Table 5.2 shows the means and standard deviations ( $\sigma$ ) for the measurement and Figure 5.3 shows distribution of the times measured.

Table 5.2 Times Measured for Web Pages to Compare the Effect of LOC

	Page 1a		Page 1b		Page 1c		Page 1d	
	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$
Rendering Time (ms)	16.1	0.57	15.3	0.48	16.1	0.57	31.3	0.48
Response Time (ms)	31.9	0.32	31.2	0.42	47.2	0.42	62.4	0.52
Start Requesting until Start Displaying (ms)	15.8	0.42	15.9	0.32	31.1	0.32	31.1	0.32

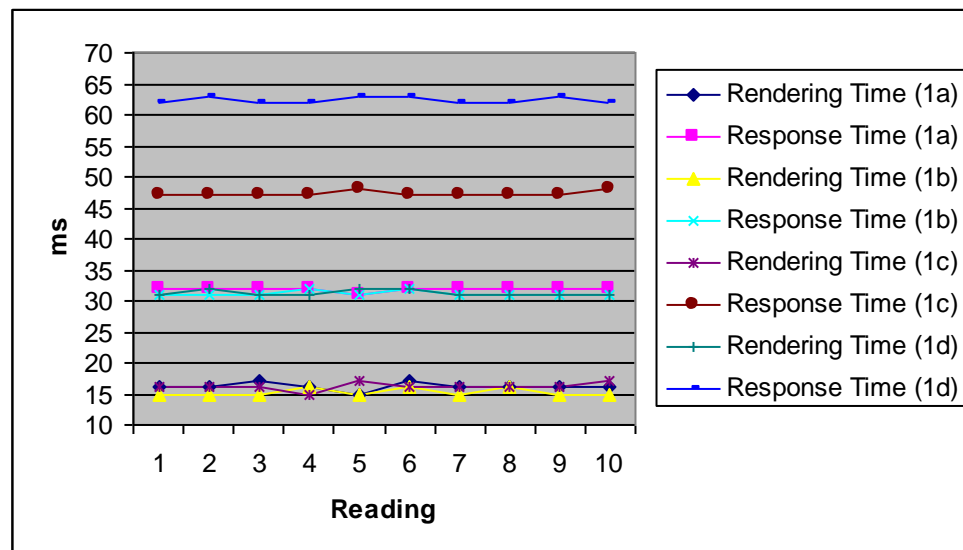


Figure 5.3 Distribution of Times Measured for Web Pages to Compare the Effect of LOC

It is obvious that there is very little difference between the times measured for Pages 1a and 1b. It is not likely to be noticed by the end users. Comparing Pages 1a and 1c where their LOC are both 108, it is found that both pages show similar rendering time and the difference in response time comes from the length of time from the moment the request was sent until the displaying of the page starts. This duration between sending the request and displaying remains consistent when the LOC changes from 108 in Page 1c to 1008 in Page 1d. Even when the LOC increases from 108 in Page 1a to 1008 in Page 1d, the response

time only increases by about 30ms. The measurements show that  $w_l$  is actually a very small value and is only affected by the capacity of the client browser to render the pages. Meanwhile, the difference in the duration between sending a request and displaying the full page mainly results from the difference in the size of the pages.

Considering the size of the pages, Pages 1a and 1b are similar while Pages 1c and 1d are close to each other. It takes, on average, about 16ms for the browser to start displaying Pages 1a and 1b from the moment the requests were sent. For Pages 1c and 1d, the duration is about 31ms. This means the server and network conditions were stable for handling of the request and transmission of the response. However, Page 1a shows longer response time than Page 1b and page 1d shows longer response time than Page 1c. This shows a long page (with more LOC) tends to have slightly longer rendering and response times than a shorter page that has the same size. Hence, the measurements prove that LOC affects Web page response time,  $R$ , and thus,  $R = f(LOC)$  holds.

### 5.5.2 Number of Embedded Objects

The second set of three static Web pages mainly differ in terms of number of embedded objects (JPEG images) they contain. Page 2a contains 5 embedded objects, Page 2b contains 10 embedded objects, and Page 2c contains 15 embedded objects. All the objects have the same resolution, i.e. 1600 x 1200 pixels. Total size of embedded objects in each page are identical, i.e. 3600 kB. Table 5.3 shows details of the pages that are used to model their complexities.

As explained in Section 5.5.1,  $w_l$  assigned to a line of code for static pages is actually a very small value. Moreover, Pages 2a, 2b, and 2c have very few and similar LOC. The total

size of the pages are identical too. Thus, any difference in response times measured can be attributed to number of embedded objects in the pages.

Table 5.3 Details for Pages Differ in Number of Embedded Objects

	Page 2a	Page 2b	Page 2c
LOC	13	18	23
Programming Constructs	0	0	0
State	0	0	0
No. of Embedded Objects	5	10	15
Total Page Size (kB)	3601	3601	3601
Elements	0	0	0

10 measurements of response time were taken for each page. Table 5.4 shows the means and standard deviations ( $\sigma$ ) for the measurements.

Table 5.4 Times Measured for Web Pages to Compare the Effect of Number of Embedded Objects

	Page 2a		Page 2b		Page 2c	
	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$
Rendering Time (ms)	922	25	1476	33	1995	37
Response Time (ms)	955	26	1527	25	2056	35
Start Requesting until Start Displaying (ms)	33	5	51	18	61	17

Not surprisingly, the measurements show that response time of a Web page is directly proportionate to the number of embedded objects it contains. This is explained by the fact that the more embedded objects in a page, the more time the browser needs to lay out each object. Increased numbers of embedded objects also results in increased communications between the client and the server, including setting up or re-establishing TCP connections between them. More intensive rendering as well as additional communication and transmission between the client and the server also yields higher standard deviations of response time. In the Internet, the increment in response time might be more significant as communication and transmission between the client and the server are more prone to the

influence of unpredictable and uncontrollable network conditions. This also implies that with more embedded objects, it becomes more difficult to guarantee QoS of a Web page in terms of its response time.

Meanwhile, the measurements show that the increment in response time is somewhat linearly correlated to the increment in number of embedded objects as can be seen in Figure 5.4. However, it is beyond the scope of this research to prove that the relationship is actually linear. It is also demonstrated that when the number of embedded objects increases, it takes longer for the Web browser to start displaying the response received from the server after the request was sent. This can be justified by the fact that the browser needs to parse the HTML content and calculate how to display it before the page is laid out (Killelea, 2002). With more embedded objects in a page, the parsing and calculating steps become more resource intensive and time consuming. As a consequence, the page will take slightly longer before its display starts.

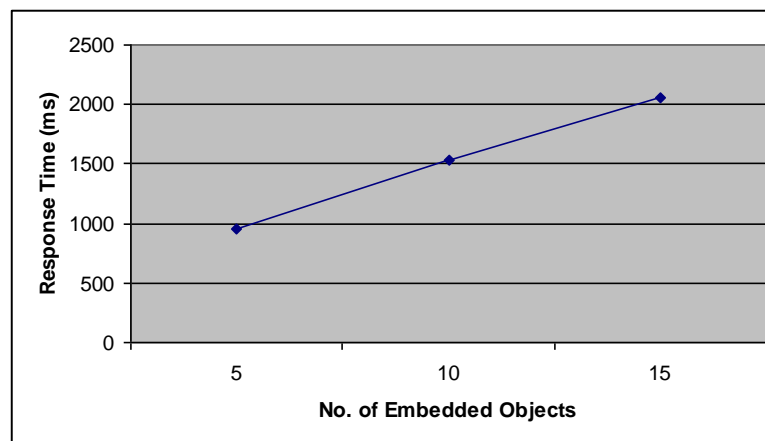


Figure 5.4 Response Time as a Function of Number of Embedded Objects

Figure 5.5 shows distribution of the measurements. The distribution exhibits a close correlation between response and rendering times for the three pages. An examination of

the correlation between the two time instances shows that the correlation coefficients are 0.982 (Page 2a), 0.852 (Page 2b), and 0.886 (Page 2c). The measurements also confirm that response time,  $R$ , of a Web page, can be expressed as a function of its number of embedded objects, i.e.  $R = f(n_o)$ .

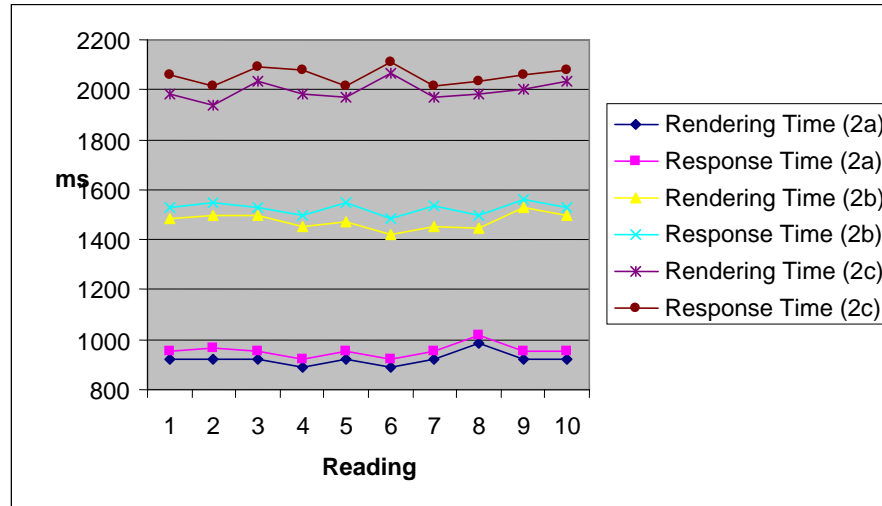


Figure 5.5 Distribution of Times Measured for Web Pages to Compare the Effect of Number of Embedded Objects

### 5.5.3 Total Size of the Page

The third set of three static Web pages differs in terms of their total size. Each page contains 5 embedded objects (JPEG images); each in turn has the resolution of 1600 x 1200 pixels. However, sizes of the embedded objects are different, resulting in different total sizes of the three pages as shown in Table 5.5.

Table 5.5 Details for Pages Differ in Total Page Size

	Page 3a	Page 3b	Page 3c
LOC	13	13	13
Programming Constructs	0	0	0
State	0	0	0
No. of Embedded Objects	5	5	5
Total Page Size (kB)	1501	2001	2501
Elements	0	0	0

Pages 3a, 3b, and 3c differ only in terms of their total size. If the total size of the page is denoted as  $s_p$ , then its content complexity can be expressed as  $C_2 = f(s_p)$ . 10 measurements of response time were taken for each page. Any difference in response times measured is attributed to  $s_p$  of the page. Table 5.6 shows the means and standard deviations ( $\sigma$ ) for the measurements.

Table 5.6 Times Measured for Web Pages to Compare the Effect of Total Page Size

	Page 3a		Page 3b		Page 3c	
	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$
Rendering Time (ms)	715	15	771	12	824	16
Response Time (ms)	751	12	806	11	860	16
Start Requesting until Start Displaying (ms)	36	8	35	7	36	5

As  $s_p$  of the page increases, its response time,  $R$ , increases as well. This is particularly due to the larger amount of data to be transmitted over the network from the server to the client. Figure 5.6 depicts the apparently linear relationship between  $R$  and  $s_p$ , though not meant to be rigorously proven in the scope of this research. In the Internet, with more data packets transmitted over the network, the influence of external factors such as network delay and packet loss becomes more significant. Eventually, the relationship between  $R$  and  $s_p$  may become exponential. Nevertheless, this shows that  $R$  can be expressed as a function of  $s_p$ , i.e.  $R = f(s_p)$ .

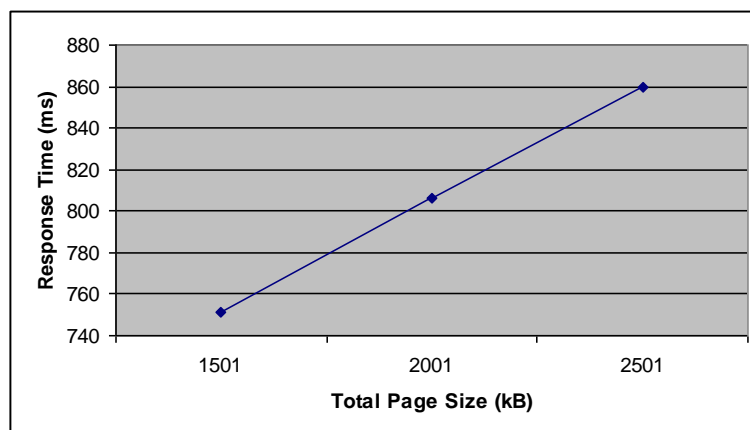


Figure 5.6 Response Time as a Function of Total Page Size

Figure 5.7 shows the distribution of the measurements. Correlation coefficients between rendering and response times for the three pages are above 0.8. This phenomenon should retain for most static pages when network transmission and server processing do not face considerable delay, such as in the LAN environment where the measurements were taken.

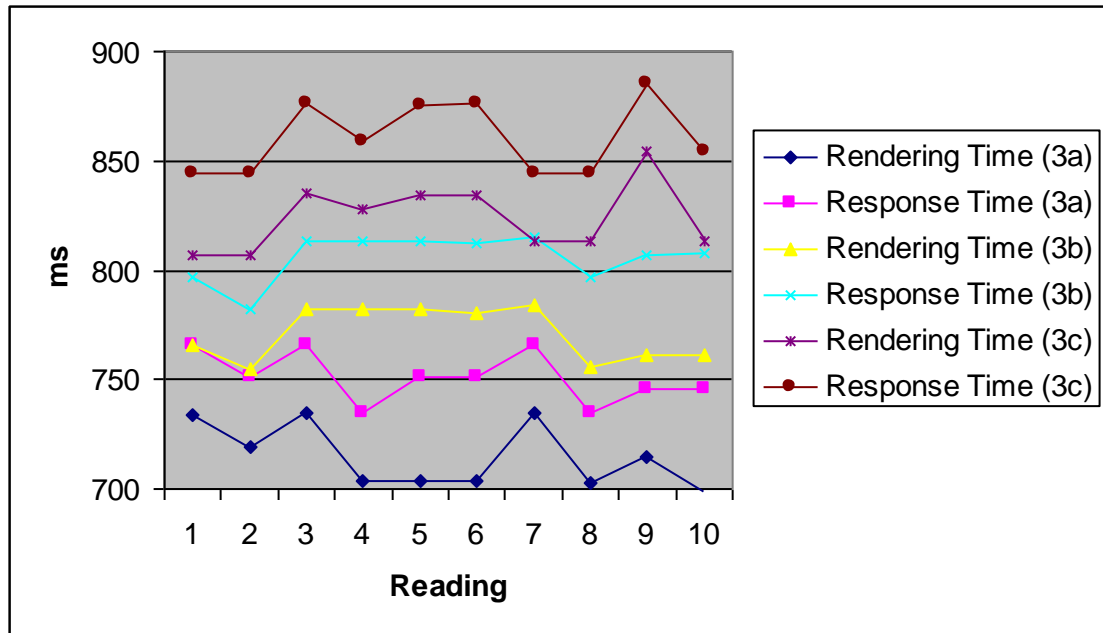


Figure 5.7 Distribution of Times Measured for Web Pages to Compare the Effect of Total Page Size

#### 5.5.4 JavaScript and CSS

Inclusion of elements, for example, JavaScripts and CSS into a Web page can cause delay in response time from two perspectives:

- The element will increase the total size of the page (See Section 5.5.3).
- The element performs certain operations that consume time contributing to the response time experienced by the users.

To show the effect of these elements, Page 1a is modified into two different pages, 1e and 1f, which include JavaScript and CSS to perform the following operations:



- Page 1e includes JavaScript that prints the current date and time at the beginning of the page.
- Page 1f includes CSS that specifies the background and font colours of the page.

Table 5.7 shows the details of the pages.

Table 5.7 Details for Pages that Include Elements

	<b>Page 1e</b>	<b>Page 1f</b>
LOC	113	114
Programming Constructs	0	0
State	0	0
No. of Embedded Objects	0	0
Total Page Size (kB)	2	2
Elements	1	1

The pages differ from Page 1a by 5 to 6 LOC and their total page sizes are larger than Page 1a by less than 1kB. 10 measurements were taken for each page and the difference in response time from Page 1a is attributed to the elements included in the page. Table 5.8 shows the means and standard deviations ( $\sigma$ ) for the measurements.

Table 5.8 Times Measured for Web Pages to Examine the Effect of JavaScript and CSS

	<b>Page 1a</b>		<b>Page 1e</b>		<b>Page 1f</b>	
	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$
Rendering Time (ms)	16.1	0.57	16.0	0.47	16.2	0.63
Response Time (ms)	31.9	0.32	46.9	0.32	47.0	0.47
Start Requesting until Start Displaying (ms)	15.8	0.42	30.9	0.32	30.8	0.42

Figure 5.8 shows the distribution of the measurements. The measurements exemplify that there is very little difference in rendering time among the three pages. The difference in response times,  $R$ , between Pages 1e and 1f with Page 1a is mainly due to the browser's pre-rendering calculation for the JavaScript and CSS to perform the necessary operation or to lay the pages out correctly. How intensively the element,  $e_r$ , needs to be processed by the

browser affects this pre-rendering time, which also comprises part of  $R$ .  $R$  is thus can be expressed as a function of  $e_r$ , i.e.  $R=f(e_r)$ .

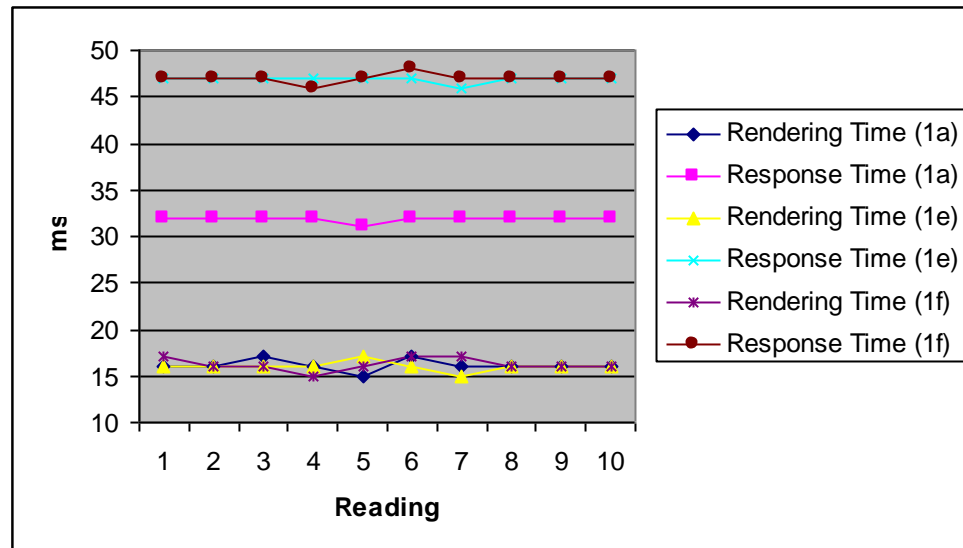


Figure 5.8 Distribution of Times Measured for Web Pages to Examine the Effect of JavaScript and CSS

### 5.5.5 Programming Constructs

The programming construct to be examined is *for* loop. Three dynamic pages are created to examine the effect of this programming construct. The pages display three drop-down boxes for three data fields of a date, i.e. day, month, and year. Three *for* loops are used to display the content for each of the three boxes. The two *for* loops to display the date and month fields are identical for the three pages. However, the *for* loop to display the year field is different among the three pages.

- Page 4a displays 100 items in the box that represents year with a single *for* loop. With Big-O notation, the complexity of this *for* loop is  $O(n)$ , with the size of  $n=100$ .
- Page 4b displays 100 items in the box that represents year with a *for* loop. Its complexity is  $O(n)$  with the size of  $n=100$ . In addition, the loop contains an inner *while* loop that iterates 500 times and performs an addition operation in each iteration. Therefore, Page 4b contains four programming constructs instead of three.

- Page 4c displays 1000 items in the box that represents year with a single *for* loop.

This means its complexity is  $O(n)$ , with the size of  $n=1000$ .

Table 5.9 shows the details of the pages.

Table 5.9 Details for Pages with Different Programming Construct Complexity

	Page 4a	Page 4b	Page 4c
LOC	50	56	50
Programming Constructs	3	4	3
State	0	0	0
No. of Embedded Objects	0	0	0
Total Page Size (kB)	6	6	21
Elements	0	0	0

10 measurements of response time were taken for each page. Any difference in response times measured can be attributed to the complexity of the constructs that generate the pages.

Table 5.10 shows the means and standard deviations ( $\sigma$ ) for the measurements.

Table 5.10 Times Measured for Web Pages to Compare the Effect of Programming Constructs

	Page 4a		Page 4b		Page 4c	
	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$
Rendering Time (ms)	71	8	71	8	86	9
Response Time (ms)	378	37	474	29	597	22
Start Requesting until Start Displaying (ms)	307	40	402	29	511	18

Since Pages 4a and 4b resulted in the same content being displayed in the same layout in the Web browser, there is not much difference in their rendering times. However, the browser takes longer to start displaying Page 4b (402ms) than Page 4a (307ms) after the request for the page was sent. This difference can be attributed to the extra programming construct in Page 4b (the *while* loop). Page 4c, on the other hand, resulted in more data being displayed in the browser and thus has higher rendering time than Pages 4a and 4b. Meanwhile, the size of  $n$  for the *for* loop in Page 4c is large enough to cause longer server

processing time as compared to the other two pages. As a result, after the request for the page was sent, there is a longer delay before the browser starts to display it. Correlation coefficients between response time and the duration between sending a request and the browser displaying the page are high. They are 0.980 (Page 4a), 0.964 (Page 4b), and 0.920 (Page 4c) respectively. This implies a very close relationship between server processing time (for the programming constructs) and the response time measured. Figure 5.9 illustrates the distribution of the measurements. Since rendering times for the three pages are very close to each other, the time difference between “start displaying” and “start requesting” for the pages, which is more significant and relevant than rendering time, is shown to illustrate and compare the effects of the different programming constructs.

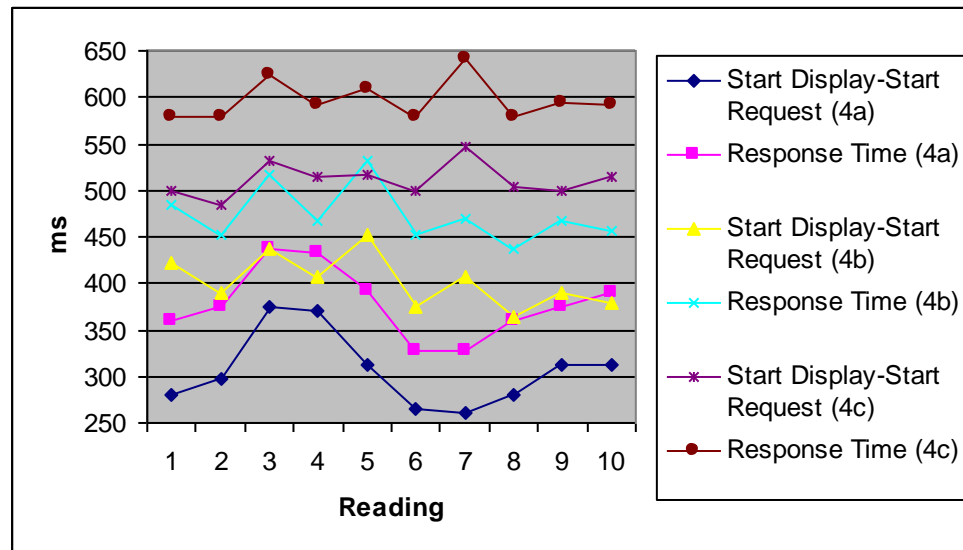


Figure 5.9 Distribution of Times Measured for Web Pages to Compare the Effect of Programming Constructs

As explained in Section 5.3, a programming construct with  $l_p$  LOC and the Big-O complexity denoted as  $O(f(n))$ , the complexity that reflects its influence on response time can be expressed as  $w_2 \times f(n) \times l_p$ , where  $w_2$  is the execution weight assigned to a line of code. Tables 5.11 shows  $f(n)$ ,  $n$  and  $l_p$  of the programming constructs in the three pages.

Table 5.11 Details of Programming Constructs in Pages 4a, 4b, and 4c

	Page 4a			Page 4b				Page 4c		
	Loop 1	Loop 2	Loop 3	Loop 1	Loop 2	Loop 3	Loop 4	Loop 1	Loop 2	Loop 3
$f(n)$	n	n	n	n	n	n	$n \times m$	n	n	n
$n$	31	12	100	31	12	100	100, 500	31	12	500
$l_p$	3	3	4	3	3	5	2	3	3	4
$f(n) \times l_p$	93	36	400	93	36	500	100000	93	36	2000

From Table 5.11, it can be found that the size of  $n$  for Loop 3 plays an important role in influencing response time. It can also be found that Loop 4 has relatively light execution weight compared to Loop 3 as it performs simple arithmetic and logical operations only. Loop 3, on the other hand, involves a Java *String* appending operation, which is more time consuming (McCluskey, 2007). This suggests that response times,  $R$ , of the pages is affected by the complexity of the programming constructs that generate them and can be expressed as a function of these programming constructs.

### 5.5.6 Database Access

The effect of database is examined through three dynamically created pages. The pages display text only in order to minimise the effect of transmission on response time.

- Generation of Page 5a involves the establishment of a database connection and query to a small table. Total size of the resulted page is 5kB that displays a drop-down box that contains 40 items.
- Generation of Page 5b involves the establishment of a database connection and two queries to two small tables. Total size of the resulted page is 6kB that displays two drop-down boxes that contains 40 and 20 items each. The difference between response times measured for Pages 5b and 5a is attributed to the extra database query, additional LOC and programming constructs to process results returned from the query, and larger resultant page to be displayed in the browser.

- Generation of Page 5c involves the establishment of two database connections and two queries to two small tables. Total size of the resulted page is 6kB that displays two drop-down boxes that contains 40 and 20 items each. The two queries and tables accessed in Page 5c are the same to those in Page 5b. Their resultant pages are identical too. Therefore, the difference between response times measured for Pages 5c and 5b is attributed to the extra database connection established.

Table 5.12 shows the details of the pages.

Table 5.12 Details for Pages with Different Database Access Attributes

	Page 5a	Page 5b	Page 5c
LOC	67	84	94
Programming Constructs	2	4	4
State (Database Access)	1 connection, 1 query	1 connection, 2 queries	2 connections, 2 queries
No. of Embedded Objects	0	0	0
Total Page Size (kB)	5	6	6
Elements	0	0	0

10 measurements of response time were taken for each page. Any difference in response times measured is mainly attributed to the difference of the number of database connections established and queries involved to generate the pages. Table 5.13 shows the means and standard deviations ( $\sigma$ ) for the measurements.

Table 5.13 Times Measured for Web Pages to Compare the Effect of Database Access

	Page 5a		Page 5b		Page 5c	
	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$
Rendering Time (ms)	44	7	70	8	70	8
Response Time (ms)	2733	39	2871	61	3092	58
Start Requesting until Start Displaying (ms)	2689	43	2801	62	3022	55

Pages 5b and 5c exhibit higher rendering times than Page 5a as they are larger. In addition, the two identical pages show the same rendering times. The difference in response times

between Pages 5a and 5b is 138ms while the difference between Page 5b and 5c is 221ms. This means opening a database connection is more time consuming than executing a query. This is supported by the fact that opening a connection to a database is a time-consuming process (Hall, 2000). The measurements also show high correlation coefficients between response time and the duration between the request was sent and the browser starts to display the page. The coefficients are 0.991 for the three pages. Figure 5.10 illustrates the distribution of the measurements.

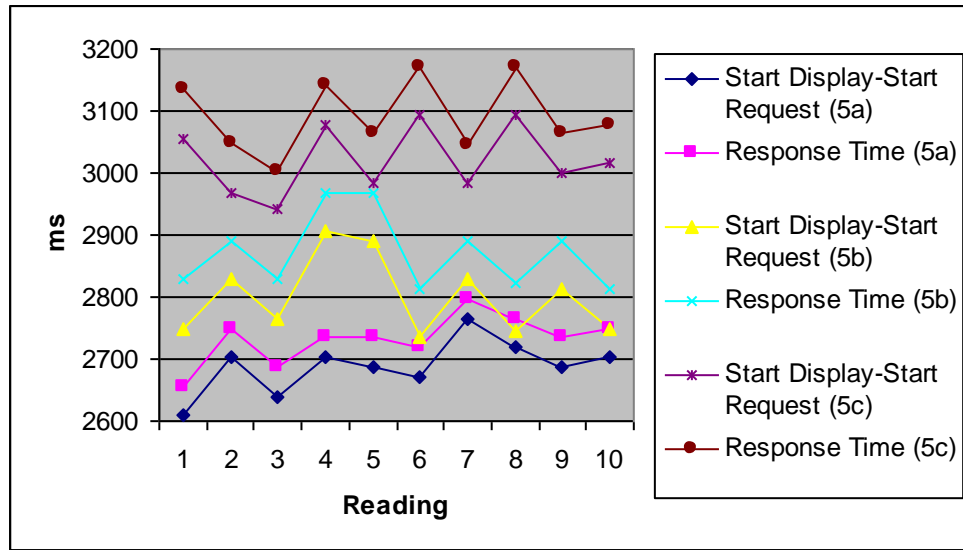


Figure 5.10 Distribution of Times Measured for Web Pages to Compare the Effect of Database Access

As described in Section 5.3.3, a Web page's creation complexity related to database is defined as  $w_3t + \sum_{i=1}^j (n_i \times w_{s_{d,i}}) + k$ , where  $w_3$  is the weight that reflects time required to establish a database connection;  $t$  is the number of connections established to  $j$  databases, each database  $d_i$  with size  $s_{d,i}$ , and  $1 \leq i \leq j$ ;  $n_i$  are number of database queries/updates;  $w_{s_{d,i}}$  reflects the time to process a query/update; and  $k$  on the other hand is an indication for the effect of session variables and cookies on response time. The measurements support the

model and show that response times,  $R$ , of the pages are affected by database accesses that were involved in their generation and can be expressed as a function of the database access.

### 5.5.7 Cookies

The effect of cookies on Web page response time is examined through four different pages. Page 6a is not associated with any cookie. Page 6b is associated with a cookie that contains 20 session variables created by the server when the page is requested. Page 6c extracts the 20 session variables contained in the cookie associated with Page 6b and modifies their values before passing them back to the client. Page 6d is similar to Page 6b except it contains 40 session variables. Pages 6b and 6d represent the case when cookies are first created and passed to the client browser. Page 6c represents the case where cookies are passed between the client browser and the server, and operations are performed on them. Table 5.14 shows the details of the pages.

Table 5.14 Details for Pages to Examine the Effect of Cookies on Response Time

	Page 6a	Page 6b	Page 6c	Page 6d
LOC	26	30	33	33
Programming Constructs	0	1	1	1
State (Cookies)	0	1 (20 variables)	1 (20 variables)	1 (40 variables)
No. of Embedded Objects	0	0	0	0
Total Page Size (kB)	1	1	1	1
Elements	0	0	0	0

10 measurements of response time were taken for each page. Table 5.15 shows the means and standard deviations ( $\sigma$ ) for the measurements while Figure 5.11 shows the distribution of the measurements.



Table 5.15 Times Measured for Web Pages to Compare the Effect of Cookies

	Page 6a		Page 6b		Page 6c		Page 6d	
	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$	Mean	$\sigma$
Rendering Time (ms)	16	0.53	16	0.42	16	0.53	16	0.52
Response Time (ms)	31	0.32	32	0.53	47	0.42	32	0.53
Start Requesting until Start Displaying (ms)	16	0.52	16	0.48	31	0.48	16	0.32

The LOC for the four pages are similar. Thus LOC's effect on the difference in response time for the pages is minimal. The programming construct in Page 6b might be sufficient to explain the small difference in response time between Pages 6a and 6b but it does not explain the difference in response time between Pages 6b and 6c. The programming constructs in both Pages 6b and 6c are *for* loops with the complexity  $O(n)$  and the maximum size of  $n$  is 20. Four statements are executed in each iteration of the *for* loops. The difference in response time between Pages 6b and 6c is best explained by the additional step of passing the cookie from the client browser to the server and the arithmetic operation performed on the variables held by the cookie before they are passed back to the client browser.

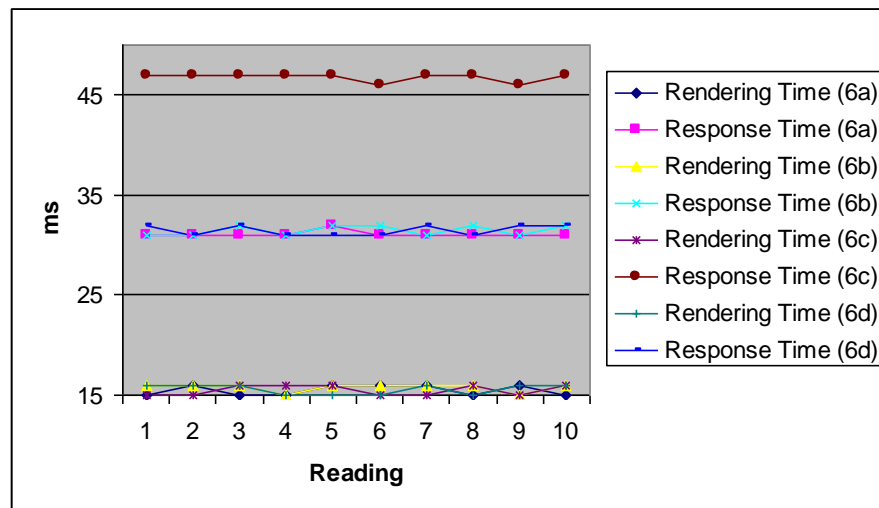


Figure 5.11 Distribution of Times Measured for Web Pages to Examine the Effect of Cookies

More session variables contained in a cookie doesn't seem to have any effect on response time. This can be shown by comparing response times for Pages 6b and 6d. In fact, a cookie by definition is just a small piece of textual information. In spite of more session variables are held, the size of a cookie is still small. For that reason, the overhead it introduces to transmission is minimal. Nevertheless, since the purpose of using cookies is to emulate an HTTP session between the client and the server, the session variables they contain are likely to be manipulated in order to maintain the session. Manipulation of the session variables in the cookies will inevitably increase response time of the Web page that the cookies are associated with. In conclusion, cookies affect response time from two aspects: transmission overhead introduced in sending them back and forth between the client browser and the server; and operations performed on them. Therefore, response time,  $R$ , of the Web page associated with cookies can be expressed as a function of the cookies and operations on the cookies involved.

## 5.6 Conclusion

Response time of a Web page can be represented by a model that reflects its complexity. Complexity of a Web page is defined in two aspects: generation ( $C_1$ ) and content ( $C_2$ ). Each of the aspect can be further elaborated into three dimensions. The three dimensions for  $C_1$  are LOC, programming constructs, and state (database access and cookies). LOC applies to both static and dynamic pages but the latter two dimensions only apply to dynamic pages. On the other hand, the three dimensions for  $C_2$  are number of embedded objects, total page size, and elements included such as JavaScript and CSS. Response times can be modelled as functions of the six complexity dimensions. Measurements were taken using the page-with-frame method described in Section 4.3 for different Web pages to examine the effect of different

complexity dimensions on response time, and to verify the models. At this stage, the models given in this Chapter treat different complexity dimensions independently. The relative importance of the dimensions in affecting Web page response time is yet to be determined. Without a precise idea of relative importance, it is extremely hard, if not impossible, to formulate a single complexity metric that combines the different dimensions to support response time modelling. Having said that, it does not mean the independent models presented in this chapter are useless. The models do provide a systematic way to check particular aspects that impact response time. Each model can be supported by a software tool that checks a Web page against the corresponding complexity dimension and thus identifies design deficiencies that cause the Web page to deliver slow responses. The implementation of the concept will be demonstrated in Chapter 7.

The next chapter explains the module that monitors response time. The monitoring tool was developed as an alternative to the measurement methods explained in Chapter 4, for estimating response time based on server access logs. Monitoring is particularly useful for observing response times of Web pages already being used in a real-life system. Chapter 7 will amalgamate the measurement, modelling, and monitoring modules together to exemplify how they could be used, as a suite of tools, to improve Web page quality in terms of response time.

## **CHAPTER 6**

### **MONITORING WEB PAGE RESPONSE TIME**

This chapter explains how Web page response time can be monitored by analysing server access log. The measurement methods explained in Chapter 4 measure response time in real-time as Web pages are downloaded, but the monitoring method outlined in this chapter determines Web page response times retrospectively based on server access log analysis. A prototype of such a monitoring system is demonstrated. Section 6.1 starts with an introduction to monitoring. Section 6.2 explains server access log. Section 6.3 describes how Web page response time can be estimated based on server log analysis. Section 6.4 shows a prototype monitoring system built based on the response time estimation method described in Section 6.3. Section 6.5 discusses results of the testing on the prototype system. Section 6.6 concludes the chapter.

#### **6.1 Monitoring**

Monitoring involves the extraction of data during program execution. Monitoring tools can be classified as pure hardware, pure software, or hybrid monitors (Haban & Wybranietz, 1990). Performance monitoring aims to achieve three goals:

- Identify performance indicators and ensure that the monitored system meets the specified performance levels in the specified context, measured based on those indicators.
- Detect and identify potential or existing problems or weaknesses in the monitored system.

- Propose actions required in order to rectify the problems or remedy the weaknesses, and thus, improve the performance of the monitored system.

Monitoring of Web sites is normally carried out in conjunction with measurement activities. One of the most common methods used for the measurement and monitoring of Web sites is active probing (see Section 2.6.1). Synthetic clients are placed in different places around the world to probe Web servers for resources hosted by the servers to monitor the servers' availability, reliability, and response time. There are many companies providing such services, e.g. Keynote (<http://www.keynote.com>), InternetSeer (<http://www.internetseer.com>), AlertSite (<http://www.alertsite.com>), Gomez (<http://www.gomez.com>), and WebMetrics (<http://www.webmetrics.com>). A common characteristic of the services provided by these companies is that the measurement and monitoring are done externally and independently of the measured/monitored servers. The measurement and monitoring results obtained are representative of much larger sets of transactions between the clients and the servers on the Internet.

The appropriateness of the results in reflecting the responses of actual Internet transactions depends on the way measurement and monitoring is carried out. Major factors that need to be considered are:

- Does the distribution of the synthetic clients reflect the distribution of actual end-users of the Web site?
- Are the Internet connection speeds of the synthetic clients well representative of the Internet connection speeds of the actual end-users?

- Do the Web browsers used reflect the proportions of different Web browsers used by the end-users?
- Is the frequency of measurements and monitoring suitable enough not to impact negatively on the performance of the Web servers being measured/monitored?
- Are the time in the day and day in the week when measurements and monitoring are carried out appropriate to depict expected access patterns?
- Do the requested resources reflect typical workloads imposed by the end-users?

Despite the limitations mentioned above, active probing still provides useful data for measurement and monitoring of Web sites in the sense that the information obtained is very similar to what might be experienced by the end-users.

However, this research adopts another commonly used monitoring method, i.e. server access log analysis (see Section 2.6.2) due to cost concerns. In contrast to active probing where typically a number of synthetic clients need to be set up separately in many geographically distributed locations, monitoring based on server access log analysis only requires a single analyser to analyse one set of server access log files. Unlike active probing, monitoring using server access log analysis does not rely on the availability of the Internet connection and does not introduce additional load to the network. Thus, server access log analysis is cheaper, easier to deploy and simpler to modify as compared to active probing.

## **6.2 Server Access Log**

Most Web servers create and maintain log files of activities performed. A server log can be created and maintained as a single file or as several files. Examples of server logs include access logs and error logs. A server access log maintains the history of page requests handled by the server. It contains information such as client IP address, request date and time, page requested, HTTP status code for the requests handled, number of bytes served, referrer, and user agent. An error log, on the other hand, records any errors that the server encountered in processing client requests as well as the corresponding error diagnostic information.

A server access log can be analysed to provide valuable information. The information includes the number of visitors to the Web site and the percentage that turned into customers that purchase items from the Web site (conversion rate), as well as keywords or advertising campaigns that brought most visitors/customers to the Web site. More confidential information, such as Web pages a user had visited in the past and the user's activities at the Web site, may be derived from server access logs. Due to this, some organisations restrict the accessibility of their server access logs to Web administrators or other key operational and administrative personnel.

Statistical analysis of server access logs may be used to provide rich information, including the following:

- Most viewed pages.
- Pages through which most visitors enter or leave the Web site.
- Web browsers used by the visitors.
- Number of visits and unique visitors to the Web site.

- Access patterns according to days of week, hours of day, etc.
- Domains or countries of the visitors.
- Keywords, phrases, and search engines that visitors used to reach the Web site.

The information is not only useful for the purpose of Web site administration but also to business related activities such as marketing, advertising, and personalisation.

A problem with developing a tool for server access log analysis to mine this data is that every server has its own format for the log files. That means that a tool needs to be tailored to suit different log formats or bespoke tools need to be built for different log formats. To overcome the problem, the World Wide Web Consortium (W3C) proposed the *Common Logfile Format* (CLF) that contains information fields as described in Table 6.1 (W3C, 1995).

Table 6.1 Common Logfile Format Information Fields

Field	Description
<i>remotehost</i>	Remote hostname (or IP number if DNS hostname is not available, or if DNSLookup is set to “Off”).
<i>rfc931</i>	The remote logname of the user.
<i>authuser</i>	The username as which the user has authenticated himself.
[ <i>date</i> ]	Date and time of the request.
" <i>request</i> "	The request line exactly as it came from the client.
<i>status</i>	The HTTP status code returned to the client.
<i>bytes</i>	The content-length of the document transferred.

With CLF, it becomes easier to write general analysers for access logs. Apart from CLF, it is also quite common for Web servers to record log files in extended CLF which captures two additional fields, i.e. *referer* and *user\_agent*. The *referer* field records the URL from which the request was made. Meanwhile, *user\_agent* shows the software, basically the Web browser, that the client used to make the request. Apart from CLF and extended CLF, some



Web servers may have proprietary log formats (Mao *et al.*, 2001), which requires custom-built server access log analysers.

On top of CLF, W3C is also working on extended log file format (W3C, 1996). The purpose of the extended log file format is to allow capturing of a wider range of data to meet the following needs:

- Permit control over the data to be recorded. A header specifying the data types recorded is written out at the start of each log. This allows a customised log file format yet it is readable by generic log file analysers.
- Support needs of proxies, clients and servers in a common format. Currently, a server has no knowledge of hits served from a proxy's cache. There is also no indication if requests came from clients behind a proxy. The extended log file format includes fields to define the parties (client, proxy, server) involved in the transactions.
- Provide robust handling of character escaping issues. In CLF, field separator character is not clearly defined and thus may occur within fields. This poses ambiguity to log file analysers. Apart from that, in order to record non-printable or non-ASCII characters, these characters are *escaped* so that they can be used in the fields, especially in a URL. One of the character escaping techniques is to replace the character with its hexadecimal value and prefix it with another character, e.g. the percent sign “%”. There is no well defined guideline for character escaping and this worsens its handling. The extended log file format defines specifications of the fields, field separator and terminator, and how they should be recorded to overcome the problem.

- Allow analysis of demographic data. A Uniform Resource Identifier (URI) for identifying HTTP sessions is proposed so that HTTP transactions can be easily grouped for demographic analysis, e.g. number of users visiting the Web site and their browsing/reading patterns. This overcomes the shortcoming of HTTP as a stateless protocol where the server views all the requests it receives as independent to each other, and thus there is no information about user sessions.
- Allow summary data to be expressed. The extended log file format introduces special fields for generating log summaries. One of the fields is *count*, which specifies the number of occurrence for a particular event. *time-from* and *time-to* are two fields to specify the *interval* that a sampling of the log file takes place.

The extended log file format supports the creation of customised log files and the customised format is readable by generic log file analysis tools. The format also contains features to support more accurate and detailed analysis. The extended log file format is still under review as of May 2007, but it can be foreseen that the realisation of the extended log file format will eventually change the landscape of Web log analysis. The discussion that follows gives a brief overview of the current state of Web log analysis.

### **6.2.1 Web Log Analysis**

Web log analysis is a common technique for mining e-commerce data (Kohavi, 2001). There are many existing Web log analysis software tools (also known as Web log analysers). Among them are Webalizer (<http://www.mrunix.net/webalizer>), AWStats (<http://awstats.sourceforge.net>), Analog (<http://www.analog.cx>), and ClickTracks (<http://www.clicktracks.com>). Generally, Web log analysers take log files as input, analyse them, and produce analysis reports as the output. The reports are made up of information

for post-mortem purposes. The information includes the number of page views within a period of time; the number of unique visitors to the Web site; domains/countries of the visitors; keywords, key phrases, and search engines used to find the Web site; most requested pages; most common entry and exit pages to the Web site; and so on. Some Web log analysers even support business-oriented analysis such as marketing campaign performance and e-commerce conversion rates.

This research, on the other hand, requires a Web log analyser that can be used for both post-mortem and real-time monitoring purposes. The analyser focuses on only one aspect: the response time. It estimates response times for Web pages requested by the clients via grouping and analysing relevant entries of a log file. The mechanism is explained in the next section.

### **6.3 Estimating Web Page Response Time Based on Server Log Analysis**

The most common use of server log analysis is to build an understanding of client activity on a Web site (Rosenstein, 2000). It is seldom used for performance monitoring for Web pages even though the task can be accomplished through incorporation of heuristic rules to reconstruct Web pages from server access log entries. Figure 6.1 shows the excerpt of a server access log in CLF. This is a short extract from an operational server access log provided by the Hunterian Museum and Art Gallery (<http://www.hunterian.gla.ac.uk/>) of the University of Glasgow. An entry in the server access log consists of information fields as described in Table 6.1. The fields are separated from each other by a white space and an entry is terminated by a Carriage Return (CR, ASCII code 13), Line Feed (LF, ASCII code 10) or CRLF. CR and LF are non-printing ASCII characters.

1	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:18 +0000] "GET
2	/museum/events/event.php?eventID=13 HTTP/1.1" 200 11210
3	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:41 +0000] "GET
4	/museum/navigation/white-theme/visit.gif HTTP/1.1" 200 1751
5	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:41 +0000] "GET
6	/museum/navigation/white-theme/services.gif HTTP/1.1" 200 1967
7	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:42 +0000] "GET
8	/museum/navigation/white-theme/whats_on.gif HTTP/1.1" 200 1766
9	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:42 +0000] "GET
10	/museum/navigation/news/bar2.gif HTTP/1.1" 200 271
11	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:42 +0000] "GET
12	/museum/navigation/news/bar1.gif HTTP/1.1" 200 73
13	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:42 +0000] "GET
14	/museum/navigation/white-theme/education.gif HTTP/1.1" 200 1733
15	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:44 +0000] "GET
16	/museum/eventsImages/cellar_boy_chardinThumb.jpg HTTP/1.1" 200 2295
17	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:44 +0000] "GET
18	/museum/navigation/news/bar3.gif HTTP/1.1" 200 330
19	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:39:55 +0000] "GET
20	/museum/navigation/white-theme/collections.gif HTTP/1.1" 200 1470
21	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:40:00 +0000] "GET
22	/museum/navigation/news/bar5.gif HTTP/1.1" 200 73
23	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:40:03 +0000] "GET
24	/favicon.ico HTTP/1.1" 302 302
25	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:40:05 +0000] "GET
26	/navigation/white-theme/collections_hover.gif HTTP/1.1" 302 302
27	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:40:05 +0000] "GET
28	/navigation/white-theme/visit_hover.gif HTTP/1.1" 302 302
29	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:40:08 +0000] "GET
30	/navigation/white-theme/service_hover.gif HTTP/1.1" 302 302
31	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:40:11 +0000] "GET
32	/navigation/white-theme/whatson_hover.gif HTTP/1.1" 302 302
33	82.200.232.156.dial.online.kz - - [23/Dec/2005:19:40:11 +0000] "GET
34	/navigation/white-theme/education_hover.gif HTTP/1.1" 302 302
35	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:07 +0000] "GET
36	/museum/events/index.php HTTP/1.1" 200 16699
37	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
38	/museum/events/news.gif HTTP/1.1" 200 1210
39	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
40	/museum/eventsImages/TheVisionOfSaintLukeThumb.jpg HTTP/1.1" 200 2452
41	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
42	/museum/eventsImages/mainhallThumb.jpg HTTP/1.1" 200 2423
43	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
44	/museum/eventsImages/spring_in_glasgowThumb.jpg HTTP/1.1" 200 3268
45	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
46	/museum/eventsImages/gravid_uterusThumb.jpg HTTP/1.1" 200 3013
47	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
48	/museum/eventsImages/chardin_lady_taking_teaThumb.jpg HTTP/1.1" 200 1769
49	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
50	/museum/eventsImages/man_makes_the_beadsThumb.jpg HTTP/1.1" 200 3067
51	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
52	/museum/eventsImages/smallest1Thumb.jpg HTTP/1.1" 200 2050
53	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
54	/museum/eventsImages/CADELL_STILL LIFE & ROSEThumb.jpg HTTP/1.1" 200 2142
55	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
56	/museum/eventsImages/snydersThumb.jpg HTTP/1.1" 200 1904
57	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
58	/museum/eventsImages/paolozzi-planeThumb.jpg HTTP/1.1" 200 1485
59	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
60	/museum/eventsImages/crm_scot_mus_revThumb.jpg HTTP/1.1" 200 3608
61	postgrad6.law.gla.ac.uk - - [23/Dec/2005:19:42:08 +0000] "GET
62	/museum/eventsImages/loane_wide2Thumb.jpg HTTP/1.1" 200 1667

Figure 6.1 Excerpt of a Server Access Log

An entry shows a request for a resource (an HTML file, its embedded objects, etc) from the server, which is called a *hit*. A request for a Web page may comprise several hits. It is also

common that hits for different pages appear as interlaced entries in the server access log. This exacerbates the difficulty of identifying and extracting entries from the server access log that make up a request for a single page. One of the ways to deal with the difficulty is based on the *remotehost* field and the types of resources requested as indicated by the *request* field.

- When requesting a page, the *remotehost* field for the client will remain unchanged.
- A page can comprise a base object, for example, an HTML file; and several embedded objects, for example, JPEG images; each is indicated by the *request* field.

Therefore, within a specified period of time, the first request from a client that qualifies as the base object will be identified. Subsequent requests from the client that qualify as embedded objects will be grouped together with the identified base object. The step will be repeated until a request for another base object from the client is encountered or the specified time period has been exceeded. Using Figure 6.1 as an example, this process is explained below:

- The client with *remotehost* field recorded as `82.200.232.156.dial.online.kz` requested a file called `event.php`. (Lines 1 & 2)
- The `.php` file extension and `?` after the requested file's name indicates that the file is a dynamic page and the parameter passed to the server to create the page was `eventID=13`. The file qualifies as a base object. (Line 2)
- The date and time of the request were recorded as `23/Dec/2005:19:39:18`. (Line 1)
- After this request, there were 16 other requests made by the same client (identified by the *remotehost* field). These are requests for `.gif`, `.jpg`, and `.ico` files, which qualify as embedded objects. (Lines 3 to 34)

- Out of the 16 requests, the last one was made by the client on 23/Dec/2005:19:40:11, which was 53 seconds after the request for the base object. (Lines 33 & 34)
- Within the 53 seconds for the 17 requests made by the client, only one base object (the first one, Lines 1 & 2) was requested. Thus, the other 16 requests (Lines 3 to 34) are considered as embedded objects for the base object.
- If the server access log is in extended CLF, the *referer* and *user agent* (Web browser type that the client used) fields can be used as additional identifiers for this grouping process to reconstruct the Web page from the requests.
- If the maximum time to download a Web page from the site is defined to be 60 seconds, the 17 requests made can be classified as the sub-requests that make up a single page.
- Thus, the analysis shows that the client (82.200.232.156.dial.online.kz) requested the page (`event.php`) and the page's response time was (at least) 53 seconds. (Excluding rendering time.)

Repeating the analysis for the next client shown in the server access log (Lines 35 to 62), it can be seen that the client (`postgrad6.law.gla.ac.uk`) requested the page (`index.php`) with 13 embedded objects, and the page's response time was 1 second. There are a few limitations when estimating response times with this method:

- Response times estimated using this method do not include DNS lookup and TCP connection establishment times.
- The time that the first request takes to arrive at the server and the last response takes to be delivered to the client are not included.

- Browser rendering time is not fully included.
- Response times are estimated with an accuracy of seconds, rather than milliseconds.

However, the limitations do not completely disqualify the method's use for monitoring purposes. First of all, the focus of the monitoring is Web pages hosted at the server. It aims to identify Web pages that are performing poorly in terms of response time. Even though DNS lookup and TCP connection establishment times are parts of Web page response time experienced by the end-users, they are out of the control of the hosting Web server and not linked to the Web page itself. Secondly, for monitoring purposes, response times estimated at 1-second granularity level is sufficient to measure responses for Web pages with long response times. After all, the purpose of the proposed monitoring is not in measuring response times accurately but rather to identify Web pages with the problem of long response times. As the access log is available for all Web servers, our method allows monitoring of Web pages with very little effort. Even the execution of the monitoring tool with an access log as input can be automated by means of cron<sup>19</sup> jobs, which makes a solution for real-time response time monitoring.

In Section 2.4, two definitions of response time are given. The definitions are reviewed below:

- (c) The time elapsed from the moment the user requests a Web page until the requested page is displayed *in its entirety* on the user's machine. (*response time*)
- (d) The time elapsed between the start of the request and beginning of the response, i.e. the page, starts displaying on the user's machine. (*responsiveness*)

---

<sup>19</sup> A time-based scheduling service in Unix and Unix-like operating systems to execute tasks periodically on a given schedule.

The first definition of response time is adopted for the two response time measurement methods explained in Chapter 4. The response time monitoring method described so far in this chapter also takes the approach that is based on the first definition of response time. It is worthwhile to consider the second definition of response time as well.

When a user requests a page from a Web site using a Web browser, the browser contacts that server to request the Web page. When the page is delivered to the browser, the browser parses and translates the code of the page before displaying it. When the browser starts displaying the page, the user notices that the server is responding and the page is being obtained. This happens even though the page is not completely displayed yet and there may still have other elements, such as images, to be obtained by the browser for the completion of the page. This is in accordance with the second definition of response time. In order to differentiate the two definitions of response time, the second one is termed as *responsiveness*.

Our tool can also be used to monitor responsiveness of Web pages. In the context of this research, responsiveness refers to *how quickly the end-users perceive that the requested page is being received after clicking on a link or typing in the URL for a Web site*. Responsiveness is monitored by checking the time interval between the first and the second requests, as recorded in the server access log. Using Figure 6.1 as an example, responsiveness for the first page (`event.php`, Line 2) requested by the user `82.200.232.156.dial.online.kz` is the difference between the requesting time for the page (`23/Dec/2005:19:39:18`, Line 1) and the time for the request that follows it (`23/Dec/2005:19:39:41`, Line 3).



This estimation reflects responsiveness because it is based on the ways a request for a page is sent and the corresponding response is generated by the Web browser. After sending the request, the Web browser will wait for the server to respond. When the response is received, the browser will parse and display it. Any embedded objects in the page will cause subsequent requests to be sent to the server to obtain the objects. Information about all the requests will be recorded in the server access log. The duration between the receipts of the first and the second requests indicates the time taken for the server to process the first request and produce the corresponding response, as well as the time for transmitting the response to the client, and for the client browser to parse and render the response before sending the second request. It also implies how long the end-user has waited after sending the request to receive the response. Therefore, if it takes a long time between the first and second requests, we can conclude that the end-user has experienced a correspondingly long waiting time, or in other words, the page is perceived to have poor responsiveness.

This waiting time and responsiveness problem may be due to server processing of the request, transmission of the response over the network, or what is less likely, slow browser rendering of the response. Even though there are a few possible reasons for poor responsiveness, the responsiveness measure proposed here may have been caused by a problem with the Web page that directly causes the delay, and which requires further investigation. The investigation may result in the redesign of the Web page to improve its responsiveness, for example, by reducing its creation and content complexities (see Sections 5.3 and 5.4). This is particularly useful for dynamic Web pages where the time interval between the first and the second requests is normally large as compared to static Web pages, and this gives us more scope for improvement.

In the server access log shown in Figure 6.1, the request made by the first client (82.200.232.156.dial.online.kz) exhibits poor responsiveness. The interval between the first and second requests is 23 seconds, which is far greater than the desirable response times (see Section 2.4). The request made by the second client (postgrad6.law.gla.ac.uk) shows much better responsiveness, i.e. 1 second.

With the estimated Web page response time and responsiveness, the monitoring tool caters for both definitions of response time as explained in Section 2.4 and reviewed earlier in this section. The estimated response time relates to definition (a) while responsiveness relates to definition (b).

## 6.4 Prototype Monitoring System

A prototype monitoring system was built in order to estimate response time and responsiveness of Web pages. Figure 6.2 shows the conceptual model of the system.

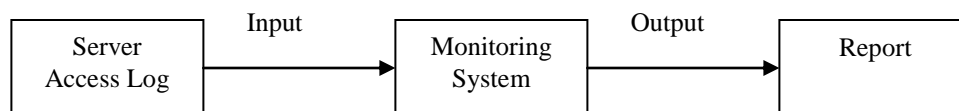


Figure 6.2 Conceptual Model of the Monitoring System

The monitoring system is built based on three assumptions:

- Each Web page consists of at least one embedded object.
- The server access log is in the CLF format.
- Grouping of the base and embedded objects used to reconstruct a Web page follows the mechanism outlined in Section 6.3.

The system was built using Java. Users choose a server access log to be analysed and the system produces the monitoring reports. An overall report shows the list of Web pages and average response times and responsiveness as well as number of requests received for each page presented as a table. The user can sort records in the table based on page name, number of requests, response time, or responsiveness. The sorting functionality makes use of the `TableSorter.java` file written by Philip Milne *et. al.* (See Appendix 4.) By clicking a record (row) in the table, the user can obtain a more detailed report for the page represented by the record. This two-level (overview, then detailed) presentation of monitoring results follows the visual design principle referred to as the *visual-information-seeking mantra* by Shneiderman (1998). The principle is summarised as “*overview first, zoom and filter, then details on demand*” (p. 523).

Apart from the information contained in the overall report, the information in the detailed reports includes median, minimum, and maximum number of embedded objects and response size for the selected page, and also the mean, median, and 90-th percentile for the response time and responsiveness of the page. The 90-th percentile for a group of values refers to the value, above which, is greater than 90% of the values in the group (Crocker & Algina, 1986). Thus, for a set of 10 values sorted in ascending order, the 90-th percentile refers to the 9th value in the list. For the number of embedded objects and response size, the median, minimum, and maximum values are chosen to illustrate the distribution of the values. For response time and responsiveness, the mean is chosen to reflect the overall performance of the Web page while the median is provided to compensate the possibility of the mean being distorted by outliers. The 90-th percentile, on the other hand, represents the 10% of worst cases of the estimated response time and responsiveness. Graphs showing

distributions of the response time and responsiveness are also presented. The structure of the reports is illustrated in Figure 6.3.

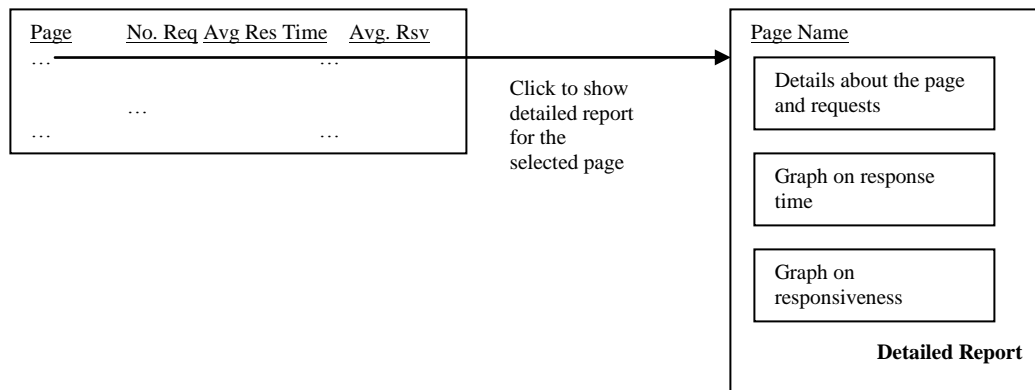


Figure 6.3 Reports for the Monitoring System

Figures 6.4 to 6.7 show user interfaces of the prototype system.

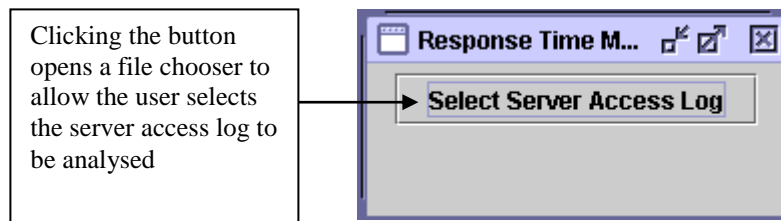


Figure 6.4 System Start-up

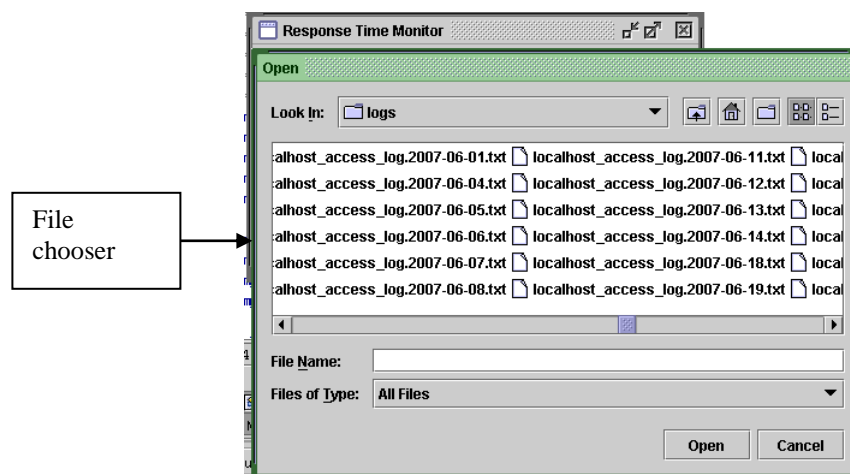


Figure 6.5 Selecting Server Access Log to be Analysed

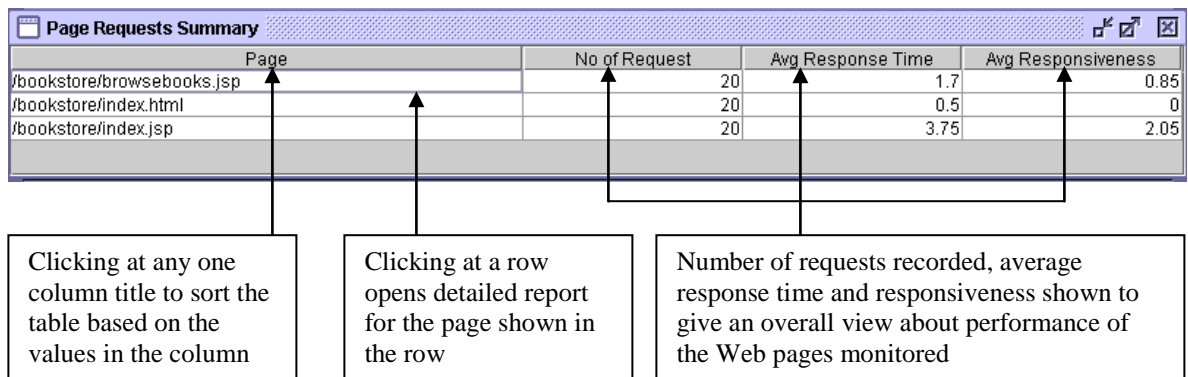


Figure 6.6 Overall Report for the Monitoring

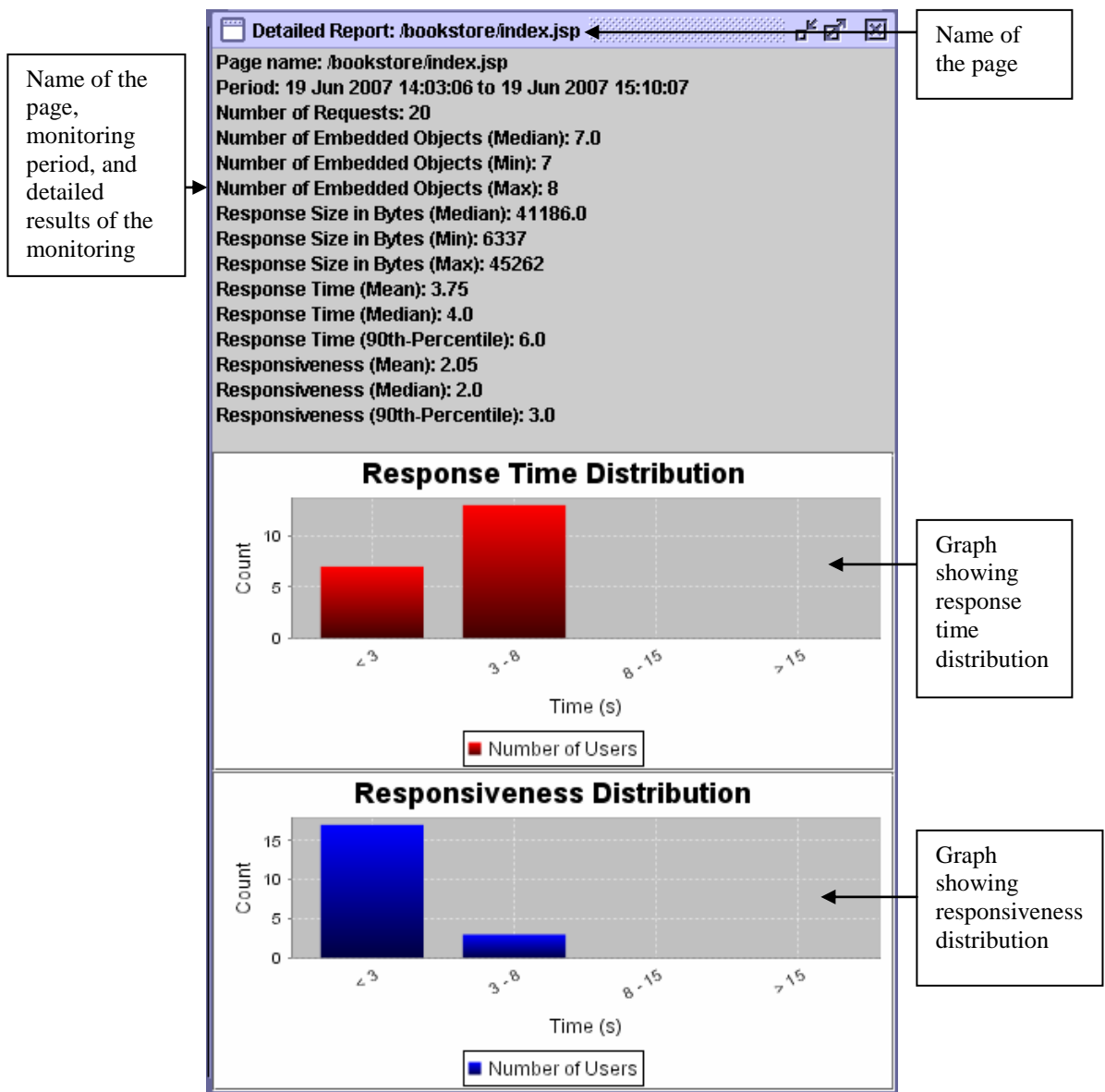


Figure 6.7 Detailed Report for a Selected Page

Source code for the system is provided in Appendix 4. The graph generation modules make use of JFreeChart, a free Java chart library for the Java™ platform, developed by Object Refinery Limited and Contributors (<http://www.jfree.org/>).

## 6.5 Testing the Prototype Monitoring System

The following steps were taken to test the system:

1. The prototype online bookstore Web site (see Section 4.4) is accessed by four different client machines on the same LAN as the Web server.
2. Each client machine requests three pages from the Web site and each page is requested five times by each client machine.
3. Response times are measured using the page-with-frames method (see Section 4.3).
4. Server access log recording the requests is analysed using the prototype monitoring system.
5. Results from the monitoring are compared with the response times measured in Step 3.

Comparison of the measured and monitored results is summarised in Table 6.2.

Table 6.2 Comparison between Measured and Monitored Results

	<b>Page</b>	<b>Measurement (s)</b>	<b>Monitoring (s)</b>
Page 1	Average response time	5.64	3.75
	Median response time	6.06	4.0
	90-th percentile response time	6.97	6.0
	Average responsiveness	2.13	2.05
	Median responsiveness	3.01	2.0
	90-th percentile responsiveness	3.36	3.0
Page 2	Average response time	0.22	0.5
	Median response time	0.19	0
	90-th percentile response time	0.39	0
	Average responsiveness	0.09	0
	Median responsiveness	0.08	0
	90-th percentile responsiveness	0.15	0
Page 3	Average response time	1.82	1.7
	Median response time	2.10	1.0
	90-th percentile response time	2.23	2.0
	Average responsiveness	1.55	0.85
	Median responsiveness	1.83	1.0
	90-th percentile responsiveness	2.05	1.0

Figures 6.8 to 6.10 are the graphs showing comparison of the results for each of the three pages.

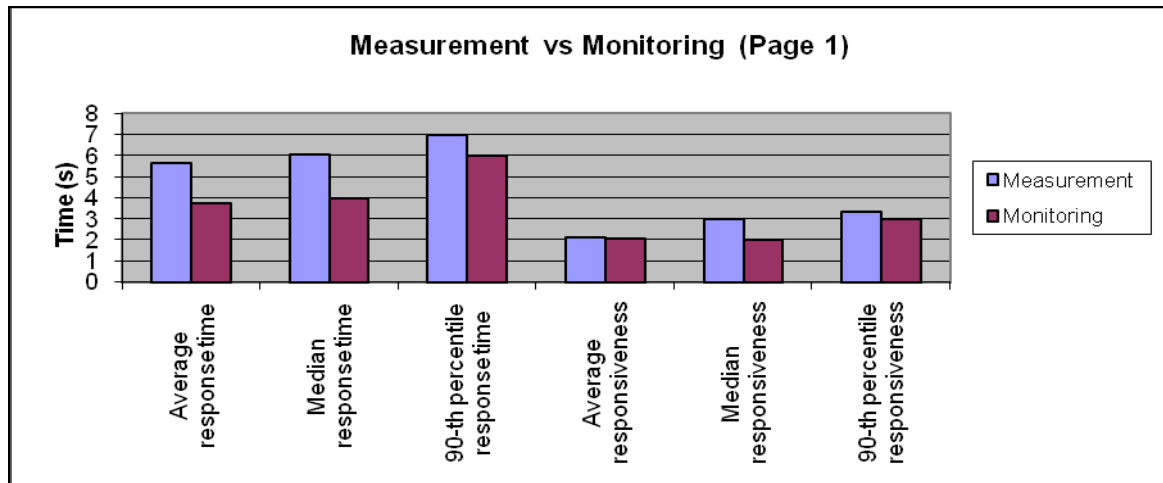


Figure 6.8 Comparison between Measured and Monitored Results for Page 1

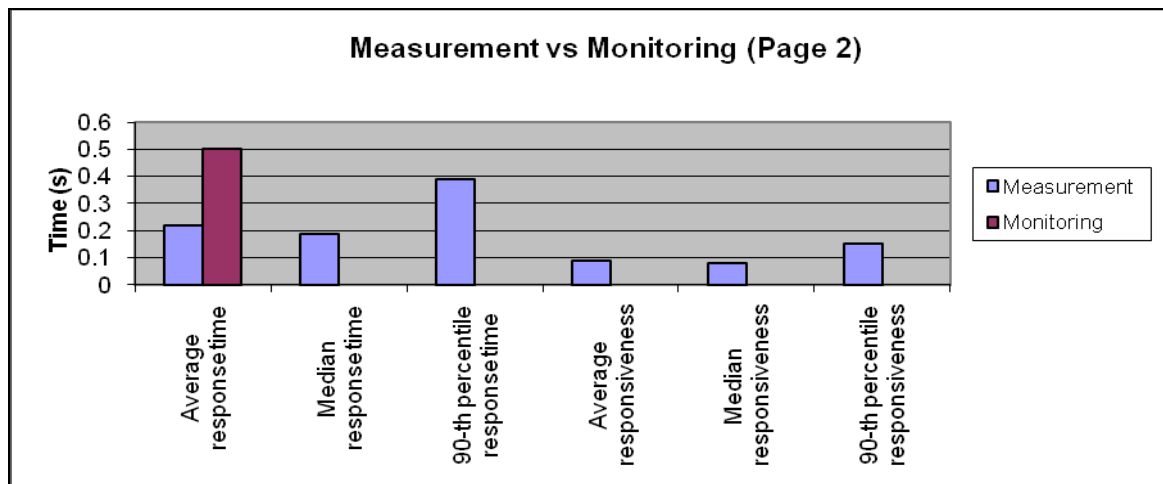


Figure 6.9 Comparison between Measured and Monitored Results for Page 2

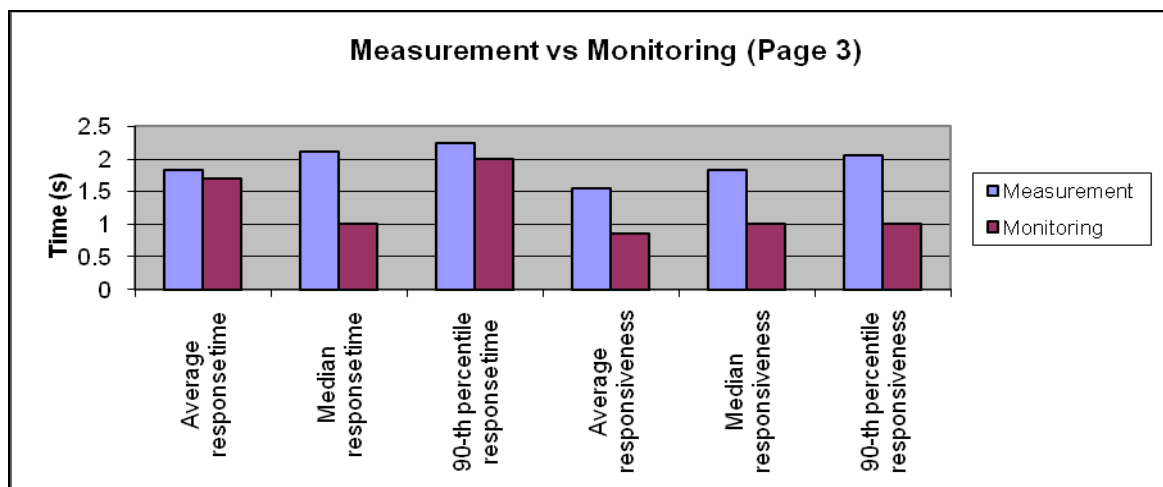


Figure 6.10 Comparison between Measured and Monitored Results for Page 3

The Server access log records the request time at the resolution of 1 second. As a result, monitoring results based on server log analysis are not as fine-grained and accurate as the results obtained with the page-with-frames method, where the times are measured at 1 millisecond granularity. The limitation is most obvious for the monitoring of Page 2 where response time and responsiveness are below 1 second. Nevertheless, the monitored results in general demonstrate similar patterns to the measured results and the monitored times are not much different from the measured times (from less than 1 second to around 2 seconds). Monitoring is therefore a useful and non-intrusive method of helping Web administrators to ascertain whether any Web pages are being inefficiently delivered to the end-users, i.e. having long response times or slow responsiveness.

## **6.6 Conclusion**

The Web page response time monitoring method presented in this chapter is an inexpensive, widely applicable, yet useful tool to help Web administrators to identify problems pertinent to particular Web page accesses at an early stage. Investigation and remedial action can be taken once problems are detected. This helps in ensuring QoS of the Web site as a whole. Ensuring satisfactory Web page response time as experienced by the end users is one of the key factors to the success of Web-based systems in general, and e-Commerce sites in particular. The prototype monitoring system proposes a way to implement the monitoring concept as well as presenting one way in which such monitoring results can be presented.

The next chapter discusses how the 3Ms (measurement, modelling, and monitoring) detailed in this and the previous two chapters can be used in the software development



lifecycle to ensure that developed Web pages deliver at least satisfactory, and, at most, minimal, response time thereby maximising the efficiency of the pages.

## CHAPTER 7

### **MAKING USE OF MEASUREMENT, MODELLING AND MONITORING TO ENHANCE WEB PAGE PERFORMANCE: AN EXAMPLE**

The three previous chapters have presented details of the Web page measurement, modelling, and monitoring (the 3Ms) tools. This chapter concludes the discussion by presenting two scenarios in which the 3Ms can be used to produce Web pages with better response time and to identify Web pages delivering less than optimal response times. The two scenarios can form part of the Web page performance analysis process, which, in turn, can form part of the quality management process during the design and maintenance phases of the software development lifecycle. This chapter is divided into four sections. Section 7.1 describes the first scenario in which measurement and modelling are used to examine Web page response time before publishing the pages. Section 7.2 illustrates the second scenario in which monitoring and modelling are used to identify Web pages that show slow response time in operation. Section 7.3 demonstrates how a proof-of-concept supporting tool can be built for Web page performance analysis. Section 7.4 concludes this chapter.

#### **7.1 Scenario 1: Designing Web Pages with Desired Response Time**

It is a distinct advantage for the response time or range of possible response time of a Web page could be estimated before the page is published to a Web server. Web pages that deliver response times longer than a specified threshold value can be scrutinised in order to improve their response time. This activity can be considered as part of a quality assurance<sup>20</sup>

---

<sup>20</sup> Quality assurance is the establishment of a framework of organisational procedures and standards which lead to high-quality software (Sommerville, 2007).

process in designing Web pages. The activity will reduce the chances of producing Web pages delivering poor response time. The measurement and modelling modules detailed in Chapters 4 and 5 respectively can be used to support this Web page response time assurance process. Figure 7.1 shows the flowchart<sup>21</sup> of this process.

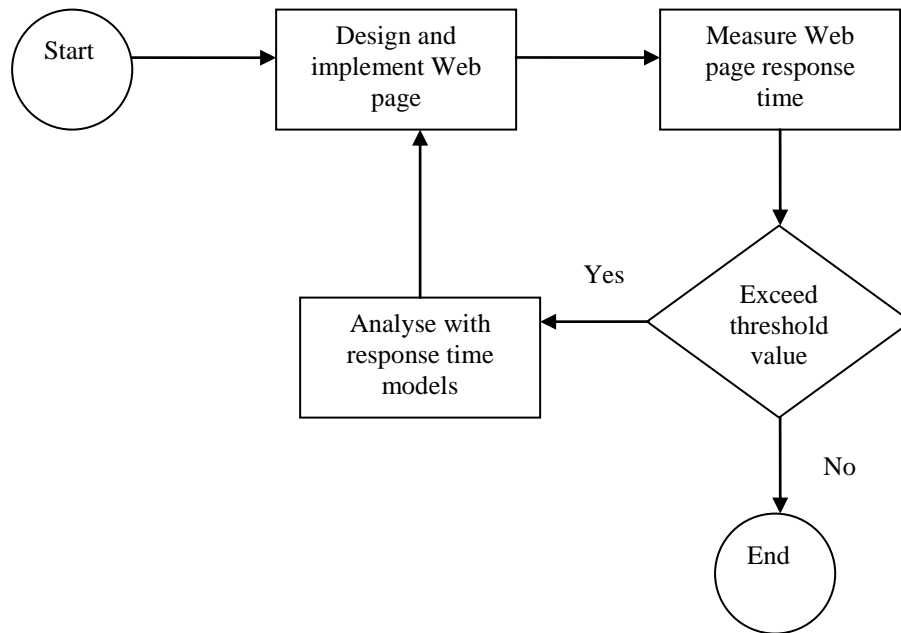


Figure 7.1 Flowchart Depicting the Process of Applying Measurement and Modelling Modules to Web Page Design

Before designing a Web page, a threshold value is set for the page to delineate the desired response time. This is a widely-accepted quality requirement<sup>22</sup> for Web pages. After the page has been designed and implemented, the page-with-frames response time measurement method described in Section 4.3 is used to measure response time. If the measured response time is below the threshold value, this indicates that the page fulfils the

<sup>21</sup> A flowchart is a graphical representation of an algorithm or a process, or the step-by-step solution of a problem, using certain special-purpose symbols such as rectangles, diamonds, ovals, and small circles.

<sup>22</sup> Quality requirements ensure a system possesses quality attributes such as usability, efficiency, reliability, maintainability, and reusability. (Lethbridge & Laganier, 2004)

desired response time requirements. Otherwise, the modelling module explained in Chapter 5 is applied to analyse the page to identify problem areas.

The following steps are taken upon analysing the page:

- (a) The Muffin proxy server method is used to measure (relative) response times for the HTML page (the base object) and its embedded objects.
- (b) If the problem is related to one or more embedded objects that take too long to download, the cause of delay in response time is likely to correspond to the content complexity outlined in Section 5.4. For each embedded object that takes too long to download, its size can be checked and, if possible, reduced. As discussed in Section 5.5.3, Web pages with larger total size take longer to download compared to Web pages that have smaller total size in terms of kilobytes. Thus, removing larger embedded objects, or replacing them with smaller objects, reduces the page response time.
- (c) If the base and embedded objects take an average time to download, but overall the page responds slowly in relation to the threshold value, this corresponds to another aspect of the content complexity, i.e. the *number* of embedded objects. As discussed in the example in Section 5.5.2, the number of embedded objects has a significant impact on response time since it takes some time for the Web browser to lay out each object on the displayed page. Moreover, communication overhead for requesting each embedded object also contributes to the page's overall response time. Take note it is assumed that transmission time over the network is consistently applied to each of the embedded objects and thus is not the factor causing the delay in the objects' response time. This problem can be resolved either through reducing the number of embedded objects or having a few Web servers to serve different

embedded objects. Another solution would be to use protocols that support persistent connection and pipelining such as HTTP/1.1.

(d) If it is the base object itself that takes too long to download, this suggests the cause probably lies in the server processing of the page, particularly the time taken to retrieve or generate the base object and deliver it to the requesting client. The cause corresponds to the *creation complexity* outlined in Section 5.3. Depending on the type (static or dynamic) of the base object, different aspects can then be examined:

- i. If the base object is a static page, the number of lines of code (LOC) it contains need to be checked. The example discussed in Section 5.5.1 shows the effect of LOC on response times for four static pages. As can be seen from the measurement results shown in Table 5.2, it can be concluded that LOC has minimal impact on Web page response time. However, if more LOC results in a lengthy page being rendered by the Web browser, it is advisable to split the page into a number of shorter pages. Splitting long Web pages not only reduces response time for an individual page, but also enhances usability (Nielsen, 2000).
- ii. If the base object is a dynamic page, then LOC, programming constructs (Section 5.5.5), database queries (Section 5.5.6), and the use of cookies (Section 5.5.7) can be checked. From the corresponding examples shown in Chapter 5, a simple guideline that can be made is to examine the programming constructs and database queries in the page more thoroughly than LOC and the use of cookies. To examine the programming constructs, algorithm analysis<sup>23</sup> can be performed to identify the possibility of modification to improve

---

<sup>23</sup> To determine how much in the way of resources, such as time or space, the algorithm will require (Weiss, 1997).

execution efficiency. Alternatively, there is the possibility of redesigning the page, e.g. splitting a page into two or more pages so that fewer or simpler programming constructs are involved in creating the page. However, this should be done without compromising the logical organisation of the page's content and logical structure of the pages. With respect to database queries, the page designer will normally have less control over the database, such as reducing the size of the database. Manipulations that the page designer *can* take include writing more efficient database queries, e.g. retrieving only the specific data that is used in the page and not all columns of a particular table.

A supporting tool for examining database queries is presented in Section 7.3.

- (e) If all the above steps do not help in reducing response time, then the page's rendering time needs to be checked, and any element included, such as JavaScript and CSS, that may cause longer rendering and thus response time can be identified. This also corresponds to content complexity. An example of the effect that inclusion of JavaScript and CSS has on Web page response time is presented in Section 5.5.4.

Figure 7.2 summarises the aforementioned steps for Web page response time analysis.

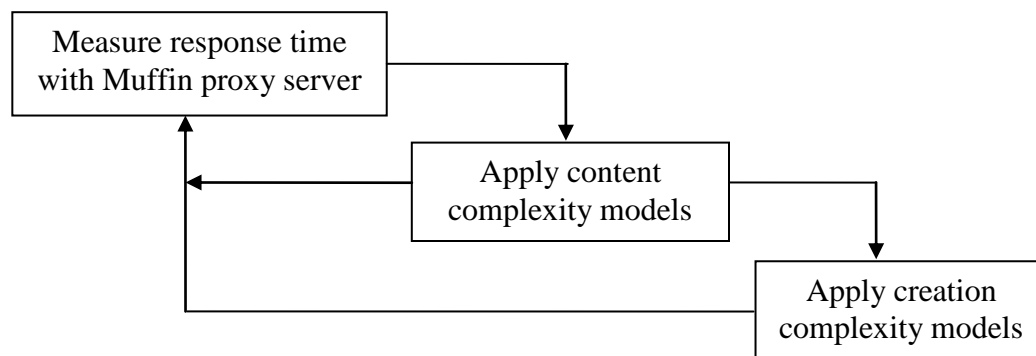


Figure 7.2 Steps for Web Page Response Time Analysis

The content complexity models are applied before the creation models for the following two reasons:

- Content complexity involves aspects that the Web page designer has more control over than creation complexity. For example, it is more feasible for the designer to reduce the number of embedded objects in a page than to reduce the size of the database or address poor table design.
- Content complexity is more relevant to *presentation* of the page, which is relatively simple to modify without adversely affecting the structure of the Web site. For example, replacing an embedded image with a lower resolution image to reduce the total page size is much easier than modifying the programming constructs for creating a dynamic Web page.

Figure 7.3 illustrates the steps in applying the content complexity models.

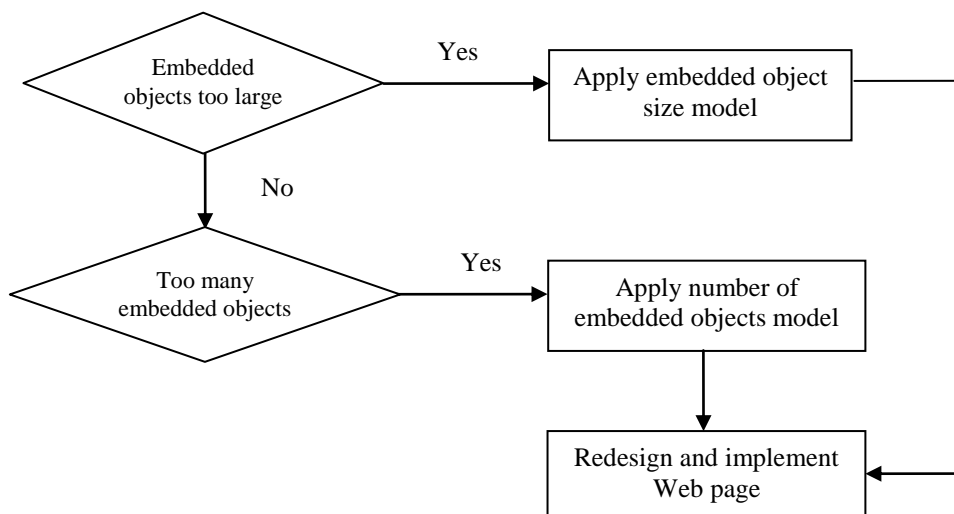


Figure 7.3 Applying Content Complexity Models for Web Page Response Time Analysis

Figure 7.4, on the other hand, illustrates the steps involved in applying the creation complexity models. Take note that content complexity and creation complexity models can be applied iteratively, in parallel or sequentially to modify the Web page until the page's response time meets the target response time requirement or is deemed satisfactory.

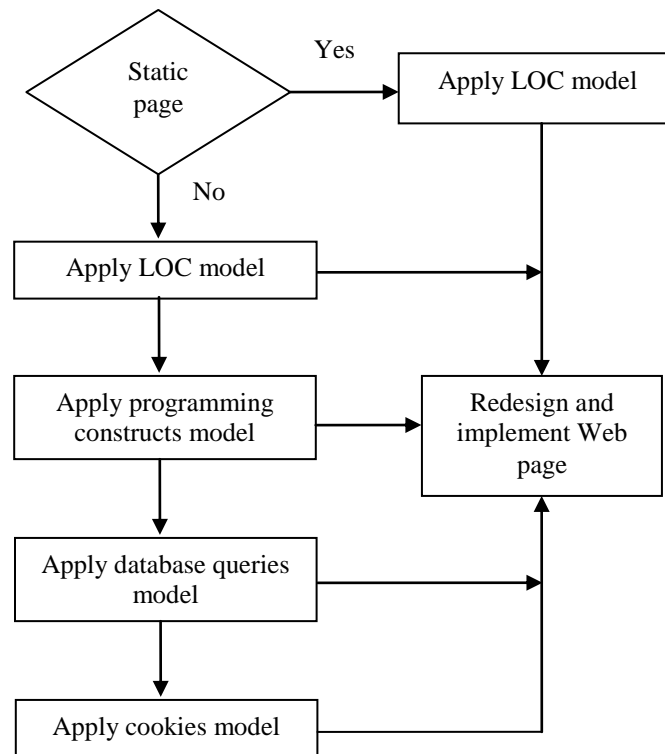


Figure 7.4 Applying Creation Complexity Models for Web Page Response Time Analysis

Applying the content complexity and creation complexity models for Web page response time analysis provides a mechanism in the Web page design process to ensure the quality of Web pages produced in terms of response time. The mechanism, as described above, proposes a systematic way of examining and improving Web page response time rather than relying on the usual trial-and-error method, or depending on the experience of Web page designer. Applying the mechanism for Web page design is in accordance with the



concept of quality assurance which requires the establishment of a framework to guide the development process in order to produce high-quality software.

## 7.2 Scenario 2: Real Time Monitoring of Web Page Response Time

Having the mechanism to support the design of Web pages with the desired response time does not guarantee the pages' *operational* response time. The slow operational response time may be due to the Web server's insufficiency in handling client requests, heavy server load, network congestion, or page design that requires further modification. Regardless of the cause, it will be an advantage to identify the set of poorly responding pages during operation and take immediate remedial action. The monitoring module outlined in Chapter 6 presents a solution to this problem. The monitoring process is depicted in Figure 7.5.

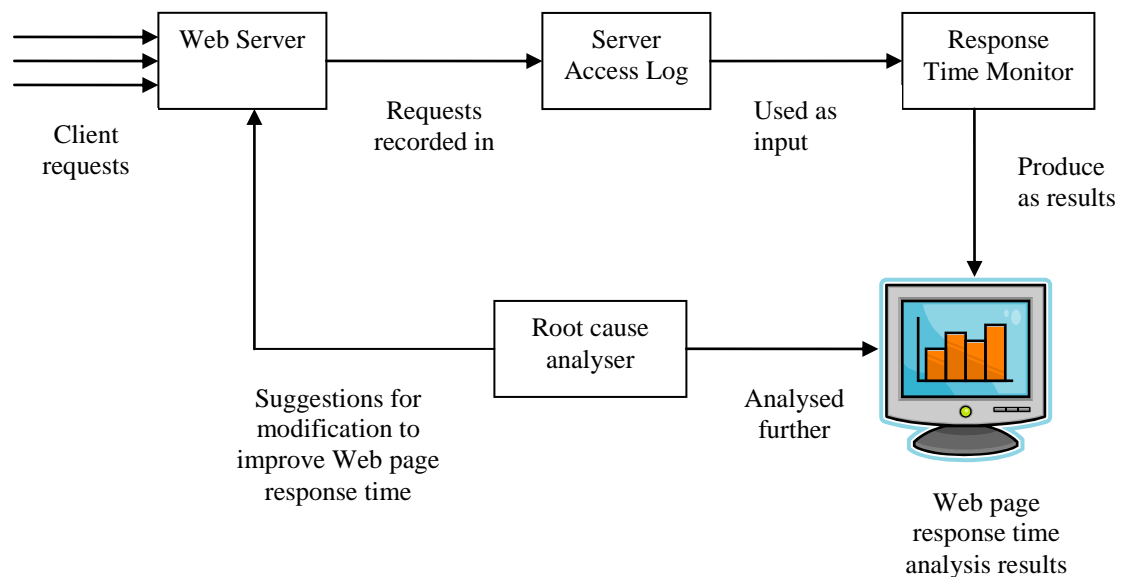


Figure 7.5 Real Time Monitoring of Web Page Response Time

Unless disabled, a server access log that records all the client requests received is available to each and every Web server. Relying on this fact, a real time monitoring mechanism can be used by any Web server as long as the server access log is available. The Web page response time monitor, as explained in Chapter 6, takes the server access log as input and analyses response time exhibited by each Web page. In order to perform real time analysis, the prototype monitoring system shown in Section 6.4 can be modified to allow the users to specify a time frame for monitoring. For example, either the previous half an hour or the previous hour. The server access log can also be offloaded to another machine so that the impact of monitoring to the Web server performance can be minimised.

As explained in Chapter 6, the monitoring tool can be used to monitor both response time and responsiveness (Section 6.3). When performing real time monitoring, all the Web pages to be monitored can be shown in a table, together with their specified threshold response time and responsiveness. The response time and responsiveness acquired from the monitoring will be included in the table too. Table 7.1 shows an example of such a table.

Table 7.1 Web Page Response Time and Responsiveness Monitoring Table

Page Name	URL	Threshold Response Time	Monitored Response Time	Threshold Responsiveness	Monitored Responsiveness	Alert
Page A	http://...	$T_{rA}$	$T_{rA(m)}$	$S_A$	$S_{A(m)}$	RT/RS
Page B	http://...	$T_{rB}$	$T_{rB(m)}$	$S_B$	$S_{B(m)}$	RT/RS
...						

Whenever Web pages exceed response times or responsiveness threshold values, the monitoring tool will generate alerts to the monitoring team. The alerts can be generated immediately when the specified response times or responsiveness are exceeded, within regular intervals of time (e.g. every 30 minutes), or upon request. The last column of Table

7.1 indicates the alert that is toggled on when response time and/or responsiveness of a Web page has been exceeded, and off when the response time and/or responsiveness is below the threshold value. The user can sort the table according to this column to easily identify Web pages that exhibit unsatisfactory response time/responsiveness and examine the cause for the delay, as well as to decide what kind of remedial action (or no action) is to be taken.

Based on the *visual-information-seeking mantra* by Shneiderman (1998), i.e. “*overview first, zoom and filter, then details on demand*” (p. 523), the monitoring tool will provide the user with detailed information about the pages upon request. The information could be presented in the form similar to Figures 6.6 and 6.7. From the detailed report, the user will ascertain the number of requests a page received, average response time and responsiveness of the page, page information such as page size and the number of embedded objects in the page, as well as the corresponding median and 90<sup>th</sup>-percentile values for response time and responsiveness, etc.

This monitoring tool provides two advantages for the Web administrator<sup>24</sup>:

- (a) To be notified within the shortest time (even nearly immediately), by means of alerts, when Web pages are responding slowly to user requests. Thus, remedial action can be taken quickly.
- (b) To be given clues about possible causes for the delay in response time or poor responsiveness. For example, when only one among all the pages was being served slowly, the cause is likely to lie within the design of the page itself. Furthermore, if the responsiveness of a page is fast but its response time is slow, the delay may due

---

<sup>24</sup> A person who maintains Web server services that allow for internal or external access to Web sites.

to the embedded objects of the page. Thus, appropriate remedial action can be taken accordingly.

In order to identify appropriate remedial action to be taken, the steps described below should be followed:

- (a) If the *responsiveness* of a page is slow, examine its *creation complexity* (Section 5.3), which includes LOC, programming constructs, database queries, and the use of cookies in the page.
- (b) If a page's *response time* is poor, examine its *content complexity* (Section 5.4), which includes the size, and number of embedded objects in the page.
- (c) Elements included in the page, such as JavaScript and CSS, can be examined in addition to the two aspects above.

Details of the steps are discussed in the previous section.

### **7.3 A Supporting Tool for Examining Database Queries in a Web Page**

Examining creation and content complexity of a Web page is a non-trivial job. For example, in order to identify how many embedded objects are there in a page, one may need to figure out how many `<object>`, `<img>`, `<applet>`<sup>25</sup>, or proprietary tags such as `<embed>`<sup>26</sup> and `<bgsound>`<sup>27</sup> there are in the page. Once the objects are identified, only their size can be checked. This is similar for examining programming constructs and database queries. Doing the examinations manually is tedious and error-prone. Developing automatic tools to support these examinations is an effective solution to the problem. This section discusses such a tool for examining database queries in a Web page in relation to response time.

---

<sup>25</sup> Deprecated.

<sup>26</sup> First introduced in Netscape 2.0.

<sup>27</sup> First introduced in Internet Explorer 3.0.

### 7.3.1 Developing the Tool

The database queries examining tool focuses on automatically extracting and analysing database functions in a page to determine whether the functions are formulated to behave efficiently from the point of view of response time. The database component was chosen instead of others here because of its well defined purpose (management of data), and limited yet straightforward activities (select, insert, update, delete, etc), which allow a higher level of automation to be achieved more easily.

Another point worth mentioning is that an automation tool for this purpose depends on the language or technology that the Web page to be examined uses. The reason for this is that automated code examination can only be done on the basis of encoding the syntax and rules of the implementation code, which is language and technology dependent. The tool discussed here was developed for the examination of Web pages written in ColdFusion Markup Language (CFML), a scripting language similar to HTML that uses tags. The use of tags in CFML, in turn, supports the automation of the tool.

In order to build a tool that examines database functions in Web pages, we must first know how database functions are written. The steps described below are followed for building the tool:

- In CFML, database functions are denoted with the `<CFQUERY>` and `<CFSTOREPROC>` tags. Both tags have the corresponding `</CFQUERY>` and `</CFSTOREPROC>` closing tags. Thus, the tool needs to identify pairs of the opening and closing tags and extract database functions between the pairs of tags for examination.

- The examination focuses on the SQL (Structured Query Language) queries performed with the `SELECT` statements, which retrieve data from databases.
- Data fields retrieved will be identified. They can be found between the `SELECT` and `FROM` keywords in SQL.
- After identifying the data fields retrieved by the SQL queries, the appearance of these data fields will be traced in the subsequent code to check if they are really used or not. This helps to detect and eliminate unnecessary interaction with databases that impacts Web page response time negatively.

This examination can help to improve the efficiency of database queries and this will have a knock-on effect on overall performance of the sites in terms of response time. This is particularly useful for data-driven Web sites where interactions with databases compose core activities of the site. The result of the examination is presented as a plain text file.

To build the database queries examining tool, the syntax of the SQL `SELECT` statements was first studied so that it can be encoded into the automated tool for the examination of the statements found in a Web page. The syntax is expressed in the Extended Backus-Naur Form (EBNF). Backus-Naur Form (BNF) was originally developed as a formal way to specify syntactic structure of programming languages in a context-free manner (Friedman *et al.*, 2001). EBNF, on the other hand, is an extension to BNF that allows optional, lists or sequences of elements that are common in the syntax of programming languages to be specified in more convenient ways than using BNF (Sethi, 1996).

The syntax of SQL `SELECT` statements is specified in the EBNF as illustrated in Figure 7.6 (Oracle, 2005). Those in bold capital letters are SQL keywords.

```
<select statement> ::=  
[subquery_factoring_clause]  
SELECT [hint]  
      [(DISTINCT | UNIQUE) | ALL]  
      select_list  
      FROM (table_reference {, table_reference}  
            | join_clause | '('join_clause')')  
      [where_clause]  
      [hierarchical_query_clause]  
      [group_by_clause]  
      [HAVING condition]  
      [model_clause]  
      [(UNION [ALL] | INTERSECT | MINUS) '('subquery')']  
      [order_by_clause]
```

Figure 7.6 SQL Select Statements' Syntax Expressed in the EBNF

The data fields (table columns) retrieved are found from the `select_list` and `subquery`, written in the part between the `SELECT` and `FROM` keywords. An examination into this part will reveal their existence. Their occurrences in the page can then be searched and the result will be presented. The tool is developed using Java and the source code is given in Appendix 5.

Given a ColdFusion file with `.cfm` extension, the tool will first identify query statements in the file. It then extracts data fields involved from the query statements. After that, the data fields are searched for their occurrences in the file where the comments are temporarily removed. The line numbers at which the data fields occur will be recorded. The report of the examination is a text file that contains all the query statements, lists all the data fields involved in the query statements, and presents the occurrences of the data fields. With the report, the users can examine the query statements and discover any data fields that are retrieved but not used in the file. Retrieving data fields that are not needed indeed is

perceived as having negative impact on the Web page's response time. Figure 7.7 shows an example of the report produced by the tool.

```
Page Name: F:\Temp\source\main.cfm:
Queries found:
<cfquery name="checkdeacon" datasource="xxxxx">          select count(*) as count
from deacon where id = '#session.id#'                    </cfquery>

<cfquery name="getpages" datasource="xxxxx"> select * from pages where
pagename='#session.thispage#' </cfquery>

<cfquery name="getann" datasource="xxxxx">  select * from announce order by number
desc </cfquery>

Number of queries: 3

Query Name: checkdeacon
Datasource: xxxxx
Data Field: *      Alias: count
Occurs at line(s): 114  171
Data Field: -
Occurs at line(s): 6

Query Name: getpages
Datasource: xxxxx
Data Field: *
Occurs at line(s):
Data Field: -
Occurs at line(s): 14
Data Field: url
Occurs at line(s): 17
Data Field: pageheading
Occurs at line(s): 18

Query Name: getann
Datasource: xxxxx
Data Field: *
Occurs at line(s):
Data Field: -
Occurs at line(s): 71  89
```

Figure 7.7 Report for Database Queries Analysis for Response Time Improvement

The report displays the file name of the page examined. It then shows all the database query statements found in the page and calculates the number of query statements found. After that, for each query with `select ... from` SQL statement, it displays the occurrences of the query name and data fields retrieved in the page by showing the line numbers where the query name and data fields occur. If a data field is indicated by a dash (-), it means the query name occurs without immediately being followed by any data field.



In the example shown in Figure 7.7, there are three queries found in the page `main.cfm`, each contains a `select ... from SQL` statement.

- For the first query (`checkdeacon`), the `count` function returns a data field called `count` and the data field occurs at lines 114 and 171. The query name itself is found at line 6, which is the position where the `<CFQUERY>` statement occurs in the file.
- The second query (`getpages`) uses the `*` symbol to select all columns from the `datasource`. It is found that two columns (`url` and `pageheading`) are used in the page.
- The third query (`getann`) also uses the `*` symbol to select all columns from the `datasource`. However, the tool didn't find occurrence of any specific data field retrieved by the query. It showed that the query name occurs two times at line 71 and line 89. Following the clue, a manual inspection found that the occurrence at line 71 relates to the `<CFQUERY>` statement while the occurrence at line 89 corresponds to the query name used in a `<CFLOOP>` statement. In the `<CFLOOP>` statement, five data fields retrieved by the query are used.

With help from the report, it can be found that the three query statements in `main.cfm` do not exhibit any clear inefficiency in terms of affecting response time.

### 7.3.2 Testing the Tool

In order to test the tool, five ColdFusion files were taken from a real running Web site for a community church. The files were examined using the tool, and then manually inspected. The results of the examinations using the tool and manual inspections were compared to check if the tool is able to identify the following aspects:

- Number of queries statements, together with the query names, and data sources where the queries were made to.
- Data fields retrieved.
- For each data field retrieved, its number of occurrence and where each occurs in the file.

The five files are included in Appendix 6.

Table 7.2 shows the overall comparison between inspections done manually and using the database queries examining tool. LOC refers to “lines of code” while “queries and data fields’ occurrences” refers to the occurrences of query names with `select ... from` SQL statements and data fields retrieved by those queries.

Table 7.2 Comparison between Automated and Manual Inspections of Web Pages

Page	LOC	Attributes	Automated	Manual
Page A	146	No. of Query	12	12
		No. of Query with <code>select ... from</code>	3	3
		Queries and Data Fields’ Occurrences	12	7
Page B	161	No. of Query	11	11
		No. of Query with <code>select ... from</code>	4	4
		Queries and Data Fields’ Occurrences	8	8
Page C	224	No. of Query	3	3
		No. of Query with <code>select ... from</code>	3	3
		Queries and Data Fields’ Occurrences	8	12
Page D	500	No. of Query	7	7
		No. of Query with <code>select ... from</code>	7	7
		Queries and Data Fields’ Occurrences	19	37
Page E	571	No. of Query	17	17
		No. of Query with <code>select ... from</code>	15	15
		Queries and Data Fields’ Occurrences	66	83

The comparison demonstrates that the maintenance tool is able to identify the number of queries in the page examined and determine how many among them contain `select ...`

from SQL statements. However, the tool is not yet able to track the occurrences of query names and data fields accurately due to the following reasons:

- The tool only differentiates between comments and other code in the page. The page content and CFML code are treated equally. Therefore, if the page contains words or characters that are identical to the query names or data fields identified, those occurrences in the page content will be counted as well. As a result, the tool will show more occurrences of query names and data fields than manual inspection. This is exhibited in the inspections of Page A. The problem can be solved by extracting only CFML code from the page before the inspection is done.
- The tool is not aware of the scope of the data fields. When a data field is referenced without being qualified by the query name, i.e. the data field is preceded by the query name followed by a period (.), the tool does not recognise its occurrence. For example, when the data fields are referenced within a `<CFLOOP>` statement, such as the third query shown in Figure 7.7. The consequence is exhibited in the inspections of Pages C, D, and E. To solve the problem, a more complicated syntactic analyser is required to make the tool aware of the context of the CFML code and the scope of the variables used.

It is beyond the scope of this study to build a complete database query analyser for CFML or other languages. However, the tool illustrates a way of automating examination of database queries that have impact on Web page response time. This helps particularly in improving the page's response time in relation to database queries. For example, inspection on Page E using the tool has discovered that the creation of the page involves 17 database queries, which might have negative impact on the page's response time (see Sections 5.3.2

and 5.5.6). Based on the report generated from the tool, a more in-depth manual inspection was carried out to examine the page, focusing on the database queries. The following deficiencies for Page E were revealed:

- Queries No. 3, No. 7, No. 12, and No. 17 use \* symbol to retrieve all the columns from a table, but only a few of them are referenced in the page. A basic guideline for making faster database queries is to reduce the amount of data processed, i.e. by selecting only necessary columns from the query. Therefore, the four queries can be modified and this will be expected to improve the page's response time.
- Queries No. 4 and No. 11 retrieve data from the same table. All the columns of the table are retrieved by Query No. 11 while Query No. 4 has an additional WHERE clause. Thus, the two queries might be consolidated, especially when database, rather the CPU speed, is the bottleneck in the page's response time.
- Queries No. 6 and No. 10 retrieve data that are not referenced at all in the page and therefore should be removed.

With the support of the tool, weaknesses in the design of database queries in Page E, which has 571 LOC, were discovered easily and quickly. This is a useful aid in facilitating the Web site maintenance process.

Figure 7.8 illustrates a proposed interface for presenting information on comparison between a set of Web pages being examined, which shows overall usage of database queries in the Web pages. By clicking at the name or URL of a Web page listed in the first column, detailed information for database queries used in the page shall be displayed as illustrated in Figure 7.9.

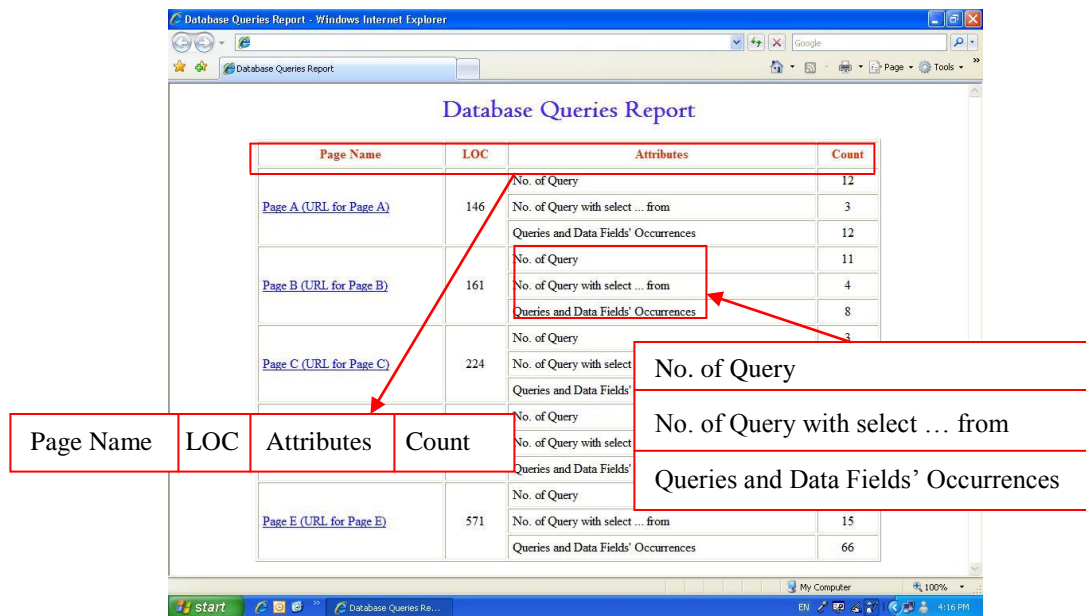


Figure 7.8 Proposed Interface for Comparison between Web Pages on the Usage of Database Queries

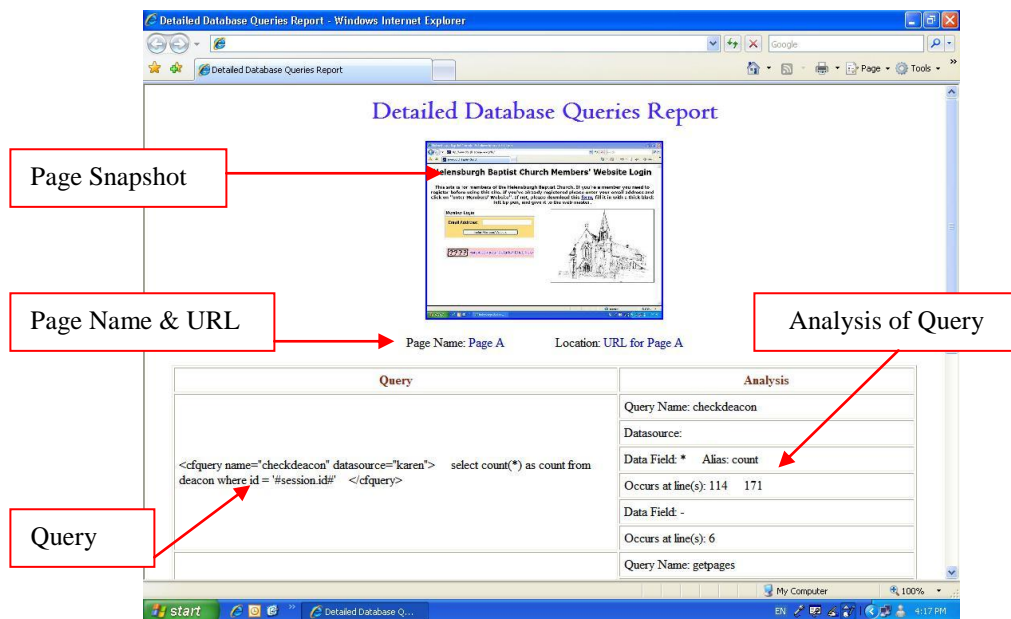


Figure 7.9 Proposed Interface for Detailed Information on the Usage of Database Queries in a Web Page

## 7.4 Conclusion

Incorporating consideration of response time into the Web page design and operation phases will improve the quality of Web pages. This chapter outlines two scenarios in which response time consideration is incorporated into the two phases:

- The first scenario shows how the measurement module (Chapter 4) and modelling module (Chapter 5) can be used in the Web page design and development process to ensure the pages exhibit more satisfactory response time.
- The second scenario shows how the monitoring module (Chapter 6) can be used during the Web page operation phase to examine responsiveness and response time of Web pages. It is also recommended that the modelling module to be used to scrutinise the design of Web pages that are found to have slow responsiveness or response time using the monitoring tool.

Tools can be built to support examination of factors that affect Web page performance in terms of response time. This chapter also describes a proof-of-concept tool that examines database queries in ColdFusion files, particularly to check if the queries are being used effectively in terms of impact on response time. It has been shown that the tool is able to produce results equivalent to manual inspection, but obviously take far less time and effort. The next chapter concludes this research by reviewing the research, discussing contributions and limitations, and proposing future work to extend the research.

## **CHAPTER 8**

### **CONCLUSION**

This chapter reviews the context, scope, and focus of this thesis. It then discusses the contributions of the research, followed by a review of the thesis statement presented in Chapter 1. Limitations of the research and future work for improvement and further research are proposed at the end of this chapter.

#### **8.1 Overview**

The thesis will be considered from three perspectives: context, scope, and focus:

##### **8.1.1 Context**

The thesis is presented in the context of the Internet, particularly the Web. The Internet has been growing exponentially since its public inception in early 1990s. The growth is evident in a few aspects: the number of Internet users, the number of Web servers connected to the Internet, the number of applications running on the Internet as their platform, and the number of services delivered over the Internet as their medium.

Their growth has resulted in various research efforts, either directly or indirectly related to the Internet. The research includes the Internet network topology, transmission protocols, server design and management, Web application design, development and maintenance, usability, user interface design and security issues, information exchange, storage and retrieval, integration of the Internet into traditionally well established fields of studies such as medicine, finance, trade and marketing, and many more. One of the fields of study that

has been attracting much research interest is performance issues for Web-based systems, which is the major scope of this thesis.

### **8.1.2 Scope**

The scope of this thesis is performance of Web pages, particularly in terms of response time. There are many metrics for illustrating performance of Web based systems, including the number of requests a Web server can handle per unit of time (throughput), the fraction of time a Web server is available to the users (availability), effectiveness for the users to accomplish tasks such as retrieving information required, and response time.

Response time for a Web-based system can be defined as the time elapsed from the moment the user requests a Web page until the requested page is displayed in its entirety on the user's machine (*response time*); or the time elapsed between the start of the request and beginning of the response, i.e. the page, starts displaying on the user's machine (*responsiveness*). Either definition quantitatively describes the users' qualitative experience in getting what they request from the Web server.

Response time can be examined from different perspectives. For example, how response times vary under different server loads or network conditions, and how to estimate the clients' Internet connection speeds in order to customise the content to be delivered to them and consequently reduce the response times the users experience. This thesis focuses on the attributes of Web pages and examines their relationship with the individual response times of Web pages.



### **8.1.3 Focus**

In contrast to some research that views response time as a function of server load or network conditions, this thesis takes a more microscopic approach by focusing on individual Web page attributes and examines the impact of the attributes on the page's response time. The attributes include the number of embedded objects in a Web page, the total size of the page, programming code, database connections and queries involved in generating the page, features such as CSS and JavaScript functions in the page, and cookies and session variables used to keep track of user sessions.

Web page structure and content can be linked to response time and this relationship is studied from three main perspectives: measurement, modelling, and monitoring (3Ms). The 3Ms constitute the overall framework and main contribution of this research.

## **8.2 Contributions**

Contributions of this thesis can be grouped based on the 3Ms mentioned in the previous section, i.e. measurement, modelling, and monitoring, as well as a proposal on making use of the 3Ms to design quality Web pages and identify Web pages that perform poorly in terms of response time or responsiveness during operational phase.

### **8.2.1 Measurement (Chapter 4)**

The thesis presents two Web page response time measurement methods: one uses a Muffin proxy server (and server access log) (Section 4.2) and another uses a Web page with two frames (Section 4.3). The method using the Muffin proxy server allows measurement of Web page response time to be carried out with some measurement overhead. The method

also allows response time for each object requested by the user to be measured. The two-frame Web page method introduces a very low measurement overhead but requires deployment at the client side. However, both methods are easy to deploy yet produce useful measurement results that help examine response times of different Web pages. The measurements can provide a quantitative understanding of the performance of Web pages in terms of their response times, or can be used to inspect the individual attributes of Web pages that affect response time.

### **8.2.2 Modelling (Chapter 5)**

The thesis presents a structured approach to modelling response time as a function of Web page complexity. Web page complexity is categorised from two perspectives: *creation* (Section 5.3) of the Web page and the page's *content* (Section 5.4). Creation refers to the ways and processes by which Web pages are created, while content refers to what Web pages are comprised of. Each of the complexity measures is further elaborated into three dimensions. The three dimensions for creation complexity are lines of code (LOC), programming constructs (decision, iteration, and function), and state (database access, session variables and cookies). LOC applies to both static and dynamic pages but the latter two dimensions only apply to dynamic pages. On the other hand, the three dimensions for content complexity are the number of embedded objects, total page size, and elements included, such as JavaScript and CSS. With this structured approach, Web page response time can be expressed as a function of the complexity dimensions, forming a solid and systematic foundation for an understanding of the relationship between attributes of Web pages and their response times.

### **8.2.3 Monitoring (Chapter 6)**

The monitoring module (Section 6.3) presented in this thesis is a widely applicable method of estimating response time experienced by end users. It relies on server access logs which are available for all Web servers. Estimation of response time is based on entries in server access logs. Estimation of response times experienced by end users helps Web administrators to identify problems pertinent to Web page accesses at an early stage. Furthermore, the estimation is done without any intrusion on the client's machine. If the estimation is done based on a server access log that is offloaded to a machine different from the Web server, it won't even deteriorate the server's performance. This thesis also proposes a possible way to implement the monitoring tool and discusses how monitoring results could be presented (Section 6.4).

Moreover, the concept of *responsiveness* (Section 6.3) is introduced in addition to *response time* (Section 2.4) to differentiate two traditional but dissimilar definitions for response time. The thesis proposes that *response time* refers to the time elapsed from the moment the user requests a Web page until the requested page is displayed *in its entirety* on the user's machine; whereas *responsiveness* refers to the time elapsed between the start of the request and beginning of the response, i.e. the page, starts displaying on the user's machine. The adoption of these two different notions should clear the ambiguity caused by different definitions of response time.

#### **8.2.4 Application of the 3Ms (Chapter 7)**

The chapter presents two scenarios which support demonstration of the application of the three afore-mentioned modules to support development and maintenance of Web pages. The first scenario shows how the measurement and modelling modules are used to design high quality Web pages (in terms of response time) before they are published (Section 7.1).

Web pages are first measured for their response time using the measurement module. Those Web pages that do not deliver the desired response time are thereby identified and further examined using the modelling module to identify causes for the exhibited poor performance. The design of the Web pages can then be improved. This gives benefit in quality assurance of the Web pages by reducing the chances of producing and publishing Web pages with poor response time.

The second scenario demonstrates how the monitoring and modelling modules are used during the operational period to quickly identify Web pages that are exhibiting slow response times or poor responsiveness (Section 7.2). The monitoring tool monitors both response time and responsiveness of Web pages being served to the users. Alerts can be triggered whenever Web pages exceed the desired response time or responsiveness specified for the pages. The Web administrator can then view the corresponding response time and responsiveness report to investigate the root cause for the delay, before deciding on what remedial action should be taken. When necessary, the modelling module can be used formally to analyse those particular Web pages. This provides an additional safe-guard step, apart from the one described in the previous scenario, to ensure that Web pages are delivered to the users within a satisfactory timeframe.

Recognising the importance of having tools to support the application of the three modules in the Web page design and operation phases, an analysis tool that examines database functions, particularly SQL queries, in Web pages is presented (Section 7.3). The tool provides an aid to the understanding of database functions in a Web page, which helps in the application of the modelling module relating to the state (database access) dimension of the creation complexity. The tool extracts database queries from the Web page to be

examined and the occurrences of data fields retrieved by the database queries in the page are tracked. With this tool, inefficiency, redundancy, and other problems related to the way database queries and result sets are used in the page can be identified and corrected. This is particularly useful in improving Web page response time since database querying is a slow process and any increase in efficiency here will impact on the response time.

### **8.3 Review of the Thesis Statement**

This section reviews the thesis statement that was presented in Section 1.1. It was claimed that it is possible to identify the contribution of *generation*, *transmission* and *rendering* to the overall Web page response time and thus identify significant cause(s) of download delay. The assertion has been proven through the measurement methods (Sections 4.2 and 4.3). By identifying contribution of generation, transmission, and rendering to the overall Web page response time, and causes of Web page download delay, remedial actions can be taken to relieve, if not resolve, the impact of the causes. The monitoring module (Section 6.3) also indicates causes to download delay by examining *response time*, *responsiveness*, and other attributes of a Web page, e.g. number of embedded objects requested.

It was also claimed in the thesis statement that it is possible to identify particular characteristics of Web pages themselves that influence response time and to provide advice to Web developers to improve the performance of these individual pages. The modelling part of this thesis (Sections 5.3 and 5.4) supports this assertion by categorising characteristics of Web pages into two types of complexities with six complexity dimensions. Web page response times are expressed as functions of the complexity

dimensions (Section 5.5) and the complexity dimensions suggest possible improvement to make Web pages to have shorter response time.

## 8.4 Limitations

The limitations of this thesis are explained below:

- The measurement method using the Muffin proxy server introduces small (but probably significant) overhead to the measurement results. This is due to the fact that the proxy server parses every request and every response and this parsing will inevitably consume time. The two-frame Web page method, on the other hand, requires direct end-user participation to make measurement possible.
- The response time model does not identify the relative importance of different complexity dimensions in affecting Web page response time.
- The accuracy of response time estimation by the monitoring module relies on the heuristics used to identify users, user sessions, and pages requested with their embedded objects. For example, the monitoring tool doesn't differentiate users behind a proxy server that appear to have the same IP address (the *remotehost* field).
- This research is yet to produce a thorough Web page design model that well incorporates consideration of response time into the design process. A case study on the successfulness of the model will be advantageous too.

## 8.5 Future Work

Future work proposed can be classified into three categories:

- a) Improvement of the 3Ms.
- b) Incorporating response time as part of the Web page design process.

- c) Extending the 3Ms into Web page maintenance process.

The three categories are discussed below.

### **8.5.1 Improvement of the 3Ms**

#### **8.5.1.1 Measurement**

The two measurement methods can be integrated and developed as an add-on to Web browsers, or even incorporated into Web browser software. In this way, actual and more accurate performance data could be collected by the server in order to improve the response times of Web pages delivered to the users. Reporting mechanisms for the measurement results, together with privacy issues, require further study.

#### **8.5.1.2 Modelling**

Investigation into the relative importance of different Web page complexity dimensions under different server and network conditions can be carried out to help to produce more accurate response-time models. These models could be applied in the design, generation, and delivery of Web pages with shorter response times. The different dimensions could be prioritised (their priority may vary under different conditions) and the Web page designers can carry out a more accurate trade-off analysis when prioritising one dimension over the others. Such decision making might well be automated at the Web server as well. For example, the Web server can choose to deliver Web pages with fewer (in terms of number) embedded objects or smaller (in terms of size) embedded objects, depending on particular server and network conditions.

#### **8.5.1.3 Monitoring**

Monitoring based on the World Wide Web Consortium (W3C)'s extended log file format proposes a better way of identifying users, user sessions, and Web pages with their embedded objects, and thus produces more accurate response time estimation. However, this future work is dependent on the realisation of the extended log file format.

#### **8.5.1.4 Supporting Tools for the 3Ms**

A set of tools can be built to support the application of 3Ms to the Web page design and operation processes. Web designers should be involved in evaluating usability of the tools. An integrated tool with good usability that facilitates the analysis of response time in the processes should be the final outcome to aim for.

#### **8.5.2 Incorporating Response Time as Part of the Web Page Design Process**

A rigorous way of incorporating response time into the design of Web pages is worth studying. It will require some well-chosen and targeted case studies to identify the most effective and efficient way to do this. A rigorous Web page design model resulting from these case studies will attribute even more value to the 3Ms proposed in this thesis.

#### **8.5.3 The 3Ms and Web Page Maintenance**

It is common for a Web page to be modified to add new functionality or new look and feel, during the course of its lifetime. It is also not unusual to use existing modules or code, probably written by somebody else, as parts of a new Web page. These are some of the common work practices that not only lead to the Web pages that are difficult to understand, but also exhibit slow response time. A reverse-engineering method that extends the concept of the 3Ms could help support maintenance of existing Web pages in order to improve their response time. The method involves extracting different components from the Web pages



to facilitate the analysis and improvement of the components, and hence, improvement of response time. In other words, a Web page can be viewed as comprising components that include:

- Page structure and layout, described using HTML and cascading style sheets (CSS).
- Page content, including text, images, multimedia elements, and hyperlinks.
- Additional functionality provided by scripting languages, e.g. JavaScript, or other Web development techniques, e.g. Ajax.
- Business and application logics reinforced by languages such as PHP, JSP, and ASP.
- Database functionality supported by languages such as SQL, PHP, JSP, and ASP.

This categorisation corresponds to Padmanabharao's (2001) view of a Web site as consisting of two main parts: the front end that the user interacts with, and everything else that includes business components and data repositories that power the Web site. Page structure and layout, content, and additional functionality correspond to the first part, while business and application logics, and database functions correspond to the second part. This modularised component-based view of a Web page is illustrated in Figure 8.1 using UML-like notations (Lethbridge & Lagniere, 2004).

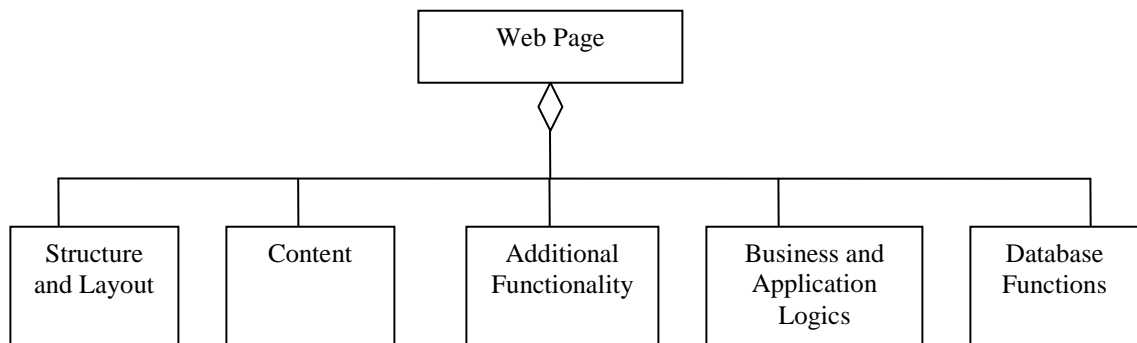


Figure 8.1 Modularised Component-based View of a Web Page

The five components can be extracted from existing Web pages and then analysed in terms of their response time. As a result, Web pages with similar or overlapping content can be consolidated; outdated Web pages can be removed; orphan pages and broken links can be identified; user interfaces can be made consistent; business and application logic can be enhanced; undocumented database table structure can be unveiled; and old, but not obsolete, data can be retrieved. The steps are useful to deliver a better user experience when browsing the Web site, including improvement in response time by means of modifying Web pages to make them respond faster.

Web page response time is of great interest to the Internet community. However, scientific research on response time is not commensurate with the interest it gains, especially research on the relationship between characteristics and response time of Web pages. The research undertaken, and reported here, has filled the gap by demonstrating how a Web page's response time could be studied in relation to its characteristics. The research has also shown that it is possible to measure, model and monitor individual Web page response time, and that doing so makes it possible to improve and optimise Web pages in terms of response time.

## REFERENCES

- Aardt, J. M. v. (2002). *Performance Measurement as a Tool for Software Engineering*. Masters Thesis, University of Pretoria, Pretoria.
- Al-Diri, K., Hobbs, D., and Qahwaji, R. (2006). Effects of Social Presence and Cultural Representation on Websites for Promoting Trust in B2C e-Commerce - A Saudi Arabian Study. In *IADIS International Conference e-Commerce 2006*, Barcelona, 9-11 December 2006.
- Ardaiz, O., Freitag, F., and Navarro, L. (2001). Estimating the Service Time of Web Clients using Server Logs. *ACM SIGCOMM Computer Communication Review*, 31(2 supplement): 108 - 123.
- Ariga, S., Nagahashi, K., Minami, M., Esaki, H., and Murai, J. (2000). Performance Evaluation of Data Transmission Using IPsec over IPv6 Networks. In *INET 2000*, Yokohama, 18-21 July 2000.
- Aweya, J., Ouellette, M., Montuno, D. Y., Doray, B., and Felske, K. (2002). An Adaptive Load Balancing Scheme for Web Servers. *International Journal of Network Management*, 12: 3-29.
- Berners-Lee, T. (1989). *Information Management: A Proposal*. Available from: <http://www.w3.org/History/1989/proposal.html> [17 September 2007].
- Bhatti, N., Bouch, A., and Kuchinsky, A. (2000). Integrating User-Perceived Quality Into Web Server Design. *Computer Networks*, 33(1-6): 1-16.
- Broadwell, P. M. (2004). Response Time as a Performability Metric for Online Services. Report No. UCB/CSD-04-1324: University of California, Berkeley.
- Box, G. E. P. (1979). Robustness in the Strategy of Scientific Model Building. In Launer, R. L. and Wilkinson, G. N. (Eds.). *Robustness in Statistics*, pp. 201-236. New York: Academic Press.
- Boyns, M. R. (2000). The Design and Implementation of a World Wide Web Filtering System. Available from: <http://muffin.doit.org/thesis/slides/> [21 March 2007].

Bucy, E. P., Lang, A., Potter, R. F., and Grabe, M. E. (1999). Formal Features of Cyberspace: Relationships between Web Page Complexity and Site Traffic. *Journal of the American Society for Information Science*, 50(13): 1246-1256.

Carver, D. L. (1988). Comparison of the Effect of Development Paradigms on Increases in Complexity. *Software Engineering Journal*, 3(6): 223-228.

Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J., and Zwaenepoel, W. (2003). Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. In *the 4th ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, 16-20, June 2003.

Cecchet, E., Marguerite, J., and Zwaenepoel, W. (2002). Performance and Scalability of EJB Applications. In *the 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Seattle, 4-8 November 2002.

Chadwick-Dias, A., McNulty, M., and Tullis, T. (2003). Web Usability and Age: How Design Changes Can Improve Performance. In *Conference on Universal Usability 2003*, Vancouver, 10-11 November 2003.

Challenger, J., Iyengar, A., Witting, K., Ferstat, C., and Reed, P. (2000). A Publishing System for Efficiently Creating Dynamic Web Content. In *IEEE Conference on Computer Communications (INFOCOM'00)*, Tel-Aviv, 26-30 March 2000.

Chandranmenon, G. P., and Varghese, G. (2001). Reducing Web Latency Using Reference Point Caching. In *the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, Anchorage, 22-26 April 2001.

Cherkasova, L., Fu, Y., Tang, W., and Vahdat, A. (2003). Measuring and Characterizing End-to-End Internet Service Performance. *ACM Transactions on Internet Technology*, 3(4): 347-391.

Chow, D. S. (2001). The Effects of Time Delay in Electronic Commerce. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, Seattle, 31 March-5 April 2001.

Clickstream. (2003). The Clickstream Advantage: Method and Benefits of Fully Automated, Multi-channel Digital Data Collection. Technical White Paper: Clickstream Technologies Plc. Available from: <http://www.clickstream.com/docs/pdf/cswitepaper.pdf> [18 March 2005].

Cohen, E., and Kaplan, H. (2000). Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency. In *IEEE Conference on Computer Communications (INFOCOM'00)*, Tel-Aviv, 26-30 March 2000.

Crocker, L., and Algina, J. (1986). *Introduction to Classical and Modern Test Theory*. New York: Harcourt Brace Jovanovich College Publishers.

Crovella, M., and Krishnamurthy, B. (2006). *Internet Measurement: Infrastructure, Traffic and Applications*. Chichester: John Wiley & Sons.

Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., and Martin, M. V. (2003). Enhancing Hyperlink Structure for Improving Web Performance. *Journal of Web Engineering*, 1(2): 93-127.

Debaraj, S., Fan, M., and Kohli, R. (2003). E-Loyalty - Elusive Ideal or Competitive Edge? *Communications of the ACM*, 46(9): 184-191.

Devillechaise, M. S., Menaud, J.-M., Muller, G., and Lawall, J. (2003). Web Cache Prefetching as an aspect: Towards a Dynamic-Weaving Based Solution. In *the 2nd International Conference on Aspect-Oriented Software Development*, Boston, Massachusetts, 17-21 March 2003.

Diao, Y., Hellerstein, J. L., Parekh, S., and Bigus, J. P. (2003). Managing Web Server Performance with AutoTune Agents. *IBM Systems Journal*, 42(1): 136-149.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol -- HTTP/1.1. RFC 2616: Network Working Group, the Internet Society.

Floyd, S., and Kohler, E. (2002). Internet Research Needs Better Models. In *the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, 28-29 October 2002.

Friedman, D. P., Wand, M., and Haynes, C. T. (2001). *Essentials of Programming Languages*, 2<sup>nd</sup> ed. Cambridge, Massachusetts: The MIT Press.

FrontRunner. (2002). FrontRunner Computer Performance Consulting. Available from: <http://www.frontrunnercpc.com> [3 November 2006].

Fujinoki, H., Sanjay, M., and Shah, C. (2003). Web File Transmission by Object Packaging - Performance Comparison with HTTP 1.0 and HTTP 1.1 Persistent Connection. In *the*

*28th Annual IEEE International Conference on Local Computer Networks*, Bonn/Königswinter, 20-24 October 2003.

Garofalakis, J., Kappos, P., and Mourloukos, D. (1999). Web Site Optimization Using Page Popularity. *IEEE Internet Computing*, July-August 1999: 22-29.

Geissler, G., Zinkhan, G. M., and Watson, R. T. (2001). Web Home Page Complexity and Communication Effectiveness. *Journal of the Association for Information Systems*, 2: 1-46.

Germonprez, M., and Zigurs, I. (2003). Causal Factors for Web Site Complexity. *Sprouts: Working Papers on Information Environments, Systems and Organizations*, 3(3): 107-121.

Greenlees, D., and Arnold, W. (2006). Asia Scrambles to Repair Quake Damage to Data Cables. *The New York Times*, 29 December 2006.

Haban, D., and Wybraniec, D. (1990). A Hybrid Monitor for Behavior and Performance Analysis of Distributed Systems. *IEEE Transactions on Software Engineering*, 16(2): 197-211.

Hall, M. (2000). *Core Servlets and JavaServer Pages*. Upper Saddle River: Prentice Hall.

Halstead, M. L. (1977). *Elements of Software Science*. New York: Elsevier North-Holland.

Hansen, W. J. (1978). Measurement of program complexity by the pair: (Cyclomatic Number, Operator Count). *ACM SIGPLAN Notices*, 13(3): 29-33.

Hitz, M., Leitner, G., and Melcher, R. (2006). Usability of Web Applications. In Kappel, G., Proll, B., Reich, S. and Retschitzegger, W. (Eds.). *Web Engineering*, pp. 219-246. Heidelberg: John Wiley & Sons.

Hoxmeier, J. A., & DiCesare, C. (2000). System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications. In *Proceedings of the 4th COLLECTeR Conference on Electronic Commerce*, Breckenridge 11 April 2000.

Iyengar, A., Nahum, E., Shaikh, A., and Tewari, R. (2002). Enhancing Web Performance. In *the 2002 IFIP World Computer Congress (Communication Systems: The State of the Art)*, Montreal, 25-30 August 2002.

Jain, R. J. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York: John Wiley & Sons.

Killelea, P. (2002). *Web Performance Tuning*. Sebastopol: O'Reilly & Associates.

Kitcharoen, S. (2007). Importance-Performance Analysis on Information Technology Applications in Higher Educational Institutions in Thailand. *ABAC Journal*, 27(2): 15-22.

Knuth, D. E. (1968). *Fundamental Algorithms*, Vol. 1. Reading: Addison-Wesley Publishing Company.

Kohavi, R. (2001). Mining e-Commerce Data: The Good, the Bad, and the Ugly. In *the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, 26-29 August 2001.

Kotsis, G. (2006). Performance of Web Applications. In Kappel, G., Proll, B., Reich, S. and Retschitzegger, W. (Eds.). *Web Engineering*, pp. 247-264. Heidelberg: John Wiley & Sons.

Krishnamurthy, B., and Wills, C. E. (2000). Analyzing Factors that Influence End-to-End Web Performance. *Computer Networks*, 33: 17-32.

Krishnamurthy, B., and Wills, C. E. (2002). Improving Web Performance by Client Characterization Driven Server Adaptation. In *Proceedings of the 11th International Conference on World Wide Web*, Honolulu, 7-11 May 2002.

Larson, K., and Czerwinski, M. (1998). Web Page Design: Implications of Memory, Structure and Scent for Information Retrieval. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Los Angeles, 18-23 April 1998.

Lethbridge, T. C., & Laganier, R. (2004). *Object-Oriented Software Engineering*, 2<sup>nd</sup> ed. Berkshire: McGraw Hill.

Leung, C. H. C. (1988). *Quantitative Analysis of Computer Systems*. Chichester: John Wiley & Sons.

Lilja, D. J. (2000). *Measuring Computer Performance: A Practitioner's Guide*. Cambridge: Cambridge University Press.

Maani, K. E., and Cavana, R. Y. (2003). *Systems Thinking and Modelling: Understanding Change and Complexity*. Auckland: Pearson Education New Zealand.

Mandel, N., and Johnson, E. J. (2002). When Web Pages Influence Choice: Effects of Visual Primes on Experts and Novices. *Journal of Consumer Research*, 29(2): 235-245.

Mao, Y., Chen, K., Wang, D., and Zheng, W. (2001). Cluster-based Online Monitoring System of Web Traffic. In *the 3rd International Workshop on Web Information and Data Management*, Atlanta, 9 November 2001.

Marshak, M., and Levy, H. (2003). Evaluating Web User Perceived Latency Using Server Side Measurements. *Computer Communications*, 26(8): 872-887.

McCabe, T. J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4): 308-320.

McCluskey, G. (2007). Java(tm) String Operations. Available from: [http://www.glenmccl.com/perfj\\_004.htm](http://www.glenmccl.com/perfj_004.htm) [17 May 2007].

Mohamed, S. S., Buhari, M. S., and Saleem, H. (2006). Performance Comparison of Packet Transmission over IPv6 Network on Different Platforms. *IEE Proceedings-Communications*, 153(3): 425- 433.

Mosberger, D., and Jin, T. (1998). httpperf - A Tool for Measuring Web Server Performance, In *the 1st Workshop on Internet Server Performance*, Madison, 23 June 1998.

Nah, F. F.-H. (2003). A Study on Tolerable Waiting Time: How Long are Web Users Willing to Wait? In *American Conference on Information Systems (AMCIS)*, Tampa, 4-6 August 2003.

Nahum, E. M., Rosu, M.-C., Seshan, S., and Almeida, J. (2001). The Effects of Wide-Area Conditions on WWW Server Performance. In *the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Cambridge, 16-20 June 2001.

Nielsen, J. (1997). The Need for Speed. Available from: <http://www.useit.com/alertbox/9703a.html> [8 February 2007].

Nielsen, J. (1999). User Interface Directions for the Web. *Communications of the ACM*, 42(1): 65-72.



Nielsen, J. (2000). *Designing Web Usability: The Practice of Simplicity*. Indianapolis: New Riders Publishing.

Oracle. (2005). Oracle Database SQL Reference 10g Release 2 (10.2), Part Number B14200-02. Available from: [http://download.oracle.com/docs/cd/B19306\\_01/server.102/b14200/statements\\_10002.htm](http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_10002.htm) [1 July 2007].

O'Reilly, T. (2005). What is Web 2.0? Design Patterns and Business Models for the Next Generation of Software. Available from: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html> [24 January 2007].

Palmer, J. W. (2002). Web Site Usability, Design, and Performance Metrics. *Information Systems Research*, 13(2): 151-167.

Padmanabharao, S. (2001). Improving User Experience through Improved Web Design and Database Performance. In S. Purba (Ed.). *High-Performance Web Databases: Design, Development, and Deployment*, pp. 623-630. Boca Raton: Auerbach Publications.

Rajamony, R., and Elnozahy, M. (2001). Measuring Client-Perceived Response Times on the WWW. In *the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, San Francisco, 26-28 March 2001.

Rosenstein, M. (2000). What is Actually Taking Place on Web Sites: e-Commerce Lessons from Web Server Log. In *the 2nd ACM Conference on Electronic Commerce*, Minneapolis, 17-20 October 2000.

Salamatian, K., and Fdida, S. (2003). A Framework for Interpreting Measurement over Internet. In *ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, Karlsruhe, 25 August 2003.

Savoia, A. (2001). Web Load Test Planning. *Software Testing and Quality Engineering*, March/April 2001: 32-37.

Sethi, R. (1996). *Programming Languages: Concepts and Constructs*, 2<sup>nd</sup> ed. Reading: Addison-Wesley.

Shi, W., Collins, E., and Karamcheti, V. (2003). Modeling Object Characteristics of Dynamic Web Content. *Journal of Parallel and Distributed Computing*, 63: 963-980.

Shneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 3<sup>rd</sup> ed. Reading, Massachusetts: Addison Wesley Longman.

Smith, W. D. (2001). TPC-W: Benchmarking an Ecommerce Solution. Technical White Paper: Intel Corporation.

SPECweb99. (2000). SPECweb99. Available from: <http://www.specbench.org/osg/web99/docs/faq.html> [7 October 2004].

Sommerville, I. (2007). *Software Engineering*, 8<sup>th</sup> ed. Essex: Pearson Education Limited.

Song, R., Liu, H., Wen, J.-R., and Ma, W.-Y. (2004). Learning Block Importance Models for Web Pages. In *the 13th International Conference on World Wide Web*, New York, 19-21 May 2004.

Stallings, W. (2005). *Business Data Communications*, 5<sup>th</sup> ed. Upper Saddle River: Pearson Education.

Stamper, D. A. (1999). *Business Data Communications*, 5<sup>th</sup> ed. Reading: Addison Wesley Longman.

Steinberg, J., and Pasquale, J. (2002). A Web Middleware Architecture for Dynamic Customization of Content for Wireless Clients. In *the 11th International World Wide Web Conference (WWW2002)*, Honolulu, 7-11 May 2002.

Strahl, R. (2000). Load Testing Web Applications Using Microsoft's Web Application Stress Tool. Available from: <http://www.west-wind.com/presentations/webstress/webstress.htm> [24 September 2004].

Titchkosky, L., Arlitt, M., and Williamson, C. (2003). A Performance Comparison of Dynamic Web Technologies. *ACM SIGMETRICS Performance Evaluation Review*, 31(3): 2-11.

W3C. (1995). Logging Control in W3C httpd. Available from: <http://www.w3.org/Daemon/User/Config/Logging.html> [13 March 2007].

W3C. (1996). Extended Log File Format: W3C Working Draft WD-logfile-960323. Available from: <http://www.w3.org/TR/WD-logfile> [13 March 2007].

Weiss, M. A. (1997). *Data Structures and Algorithm Analysis in C*, 2<sup>nd</sup> ed. Menlo Park: Addison Wesley Longman.

Wilde, E. (1999). *Technical Foundations of the World Wide Web*. Heidelberg: Springer-Verlag.

Wills, C. E., and Mikhailov, M. (2000). Studying the Impact of More Complete Server Information on Web Caching. In *the 5th International Web Caching and Content Delivery Workshop*, Lisbon, 22-24 May 2000.

Zona. (1999). *The Need for Speed, July 1999*. Research Report: Zona Research.

## BIBLIOGRAPHY

Apache. (2005). Apache HTTP Server Version 2.0 Documentation. Available from: <http://httpd.apache.org/docs/2.0/> [25-10-2005].

Barford, P., and Crovella, M. (1999). Measuring Web Performance in the Wide Area. *ACM SIGMETRICS Performance Evaluation Review*, 27(2): 37-48.

Berry, P. M. (2007). The Domain Name System. Available from: <http://www.ai.sri.com/~berry/PAM/PAM-1dom.html> [7 February 2007].

Byrne, M. D., John, B. E., Wehrle, N. S., and Crow, D. C. (1999). The Tangled Web We Wove: A Taskonomy of WWW Use. In *SIGCHI Conference on Human Factors in Computing Systems: The CHI is the Limit*, Pittsburgh, 15-20 May 1999.

Cao, P., and Liu, C. (1998). Maintaining Strong Cache Consistency in the World Wide Web. *IEEE Transactions on Computers*, 47(4): 445-457.

Cao, Y., and Ozsu, M. T. (2002). Evaluation of Strong Consistency Web Caching Techniques. *World Wide Web: Internet and Web Information Systems*, 5: 95-123.

Capilla, R., and Duenas, J. C. (2003). Light-weight Product-Lines for Evolution and Maintenance of Web Sites. In *the 7th European Conference on Software Maintenance and Reengineering (CSMR'03)*, Benevento, 26-28 March 2003.

Ceri, S., Fraternali, P., and Bongio, A. (2000). Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. *Computer Networks*, 33(1-6): 137-157.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. (2003). *Designing Data-Intensive Web Applications*. San Francisco: Morgan Kaufmann.

Challenger, J., Iyengar, A., Dantzig, P., Dias, D., and Mills, N. (2001). Engineering Highly Accessed Web Sites for Performance. In Murugesan, S. and Deshpande Y. (Eds). *Web Engineering: Managing Diversity and Complexity of Web Application Development*, Lecture Notes in Computer Science Series 2016, pp. 247-265. New York: Springer-Verlag.

Challenger, J. R., Dantzig, P., Iyengar, A., Squillante, M. S., and Zhang, L. (2004). Efficiently Serving Dynamic Data at Highly Accessed Web Sites. *IEEE/ACM Transactions on Networking*, 12(2): 233-246.

Charzinski, J. (2001). Web Performance in Practice – Why We are Waiting. *AEÜ – International Journal of Electronics and Communications*, 55(1): 37-45.

Chiew, T. K., and Renaud, K. (2005). Disassembling Web Site Response Time. In *the 12th European Conference on Information Technology Evaluation (ECITE 2005)*, Turku, 29-30 September 2005.

Chiew, T. K., and Renaud, K. (2006). Web Page Response Time as a Function of Page Complexity. In *IADIS International Conference e-Commerce 2006*, Barcelona, 9-11 December 2006.

Christopoulou, E., Garofalakis, J., Markis, C., Panagis, Y., Psaras-Chatzigeorgiou, A., Sakkopoulos, E., and Tsakalidis, A. (2003). Techniques and Metrics for Improving Website Structure. *Journal of Web Engineering*, 2(1): 90-104.

Chung, S., and Lee, Y.-S. (2000). Reverse Software Engineering with UML for Web Site Maintenance. In *the 1st International Conference on Web Information Systems Engineering (WISE'00)*, Hong Kong, 19-21 June 2000.

Cohen, E., and Kaplan, H. (2003). Proactive caching of DNS Records: Addressing a Performance Bottleneck. *Computer Networks*, 41(6): 707-726.

Coulouris, G., Dollimore, J., and Kindberg, T. (2001). *Distributed Systems: Concepts and Design*, 4<sup>th</sup> ed. Harlow: Addison Wesley.

Dobroth, K., McInerney, P., and Smith, S. (2000). Organizing Web Site Information: Principles and Practical Experience. *ACM SIGCHI Bulletin*, 32(1): 23-26.

Dyson, P., and Longshaw, A. (2004). *Architecting Enterprise Solutions: Patterns for High-Capability Internet-based Systems*. Chichester: Wiley.

Ebner, A., Proll, B., and Werthner, H. (2006). Operation and Maintenance of Web Applications. In G. Kappel, B. Proll, S. Reich & W. Retschitzegger (Eds.). *Web Engineering*, pp. 155-170. Heidelberg: John Wiley & Sons.

Fan, L., Cao, P., Lin, W., and Jacobson, Q. (1999). Web Prefetching between Low-Bandwidth Clients and Proxies: Potential and Performance. *ACM SIGMETRICS Performance Evaluation Review*, 27(1): 178-187.

Floyd, C. (2004). TPC-W: Replication of the Project. E-mail. [11-10-2004].

Garzotto, F., and Paolini, P. (1993). HDM - A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems*, 11(1): 1-26.

German, R. (2000). *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets*. Chichester: John Wiley & Sons.

Gray, N. (2003). *Web Server Programming*. Chichester, West Sussex: John Wiley & Sons.

Gschwind, T., Eshghi, K., Garg, P. K., and Wurster, K. (2002). WebMon: A Performance Profiler for Web Transactions. In *the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02)*, Newport Beach, 26-28 June 2002

Hamilton, G., Cattell, R., and Fisher, M. (1997). *JDBC Database Access with Java: A Tutorial and Annotated Reference*. Reading: Addison-Wesley.

Harrison, P. G., and Patel, N. M. (1993). *Performance Modelling of Communication Networks and Computer Architectures*. Wokingham: Addison-Wesley Publishing Company.

Haverkort, B. R. (1998). *Performance of Computer Communication Systems: A Model-Based Approach*. Chichester: John Wiley & Sons.

Haverkort, B. R., Marie, R., Rubino, G., and Trivedi, K. (2001). *Performability Modelling: Techniques and Tools*. Chichester: John Wiley & Sons.

Hofmann, R., Klar, R., Mohr, B., Quick, A., and Siegle, M. (1994). Distributed Performance Monitoring: Methods, Tools and Applications. *IEEE Transactions on Parallel and Distributed Systems*, 5(6): 585-598.

Iwaarden, J. v., Wiele, T. v. d., Ball, L., and Millen, R. (2003). Applying SERVQUAL to Web Sites: An Exploratory Study. *International Journal of Quality and Reliability Management*, 20(8): 919-935.

Iyengar, A., and Challenger, J. (1997). Improving Web Server Performance by Caching Dynamic Data. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, 8-11 December 1997.

Jansen, B. J. (2000). The Effect of Query Complexity on Web Searching Results. *Information Research*, 6(1).

Jeh, G., and Widom, J. (2003). Scaling Personalized Web Search. In *the 12th International Conference on World Wide Web*, Budapest, 20-24 May 2003.

Kamra, A., Misra, V., and Nahum, E. (2004). Controlling the Performance of 3-tiered Web Sites: Modeling, Design and Implementation. In *Joint International Conference on Measurement and Modeling of Computer Systems*, New York, 12-16 June 2004.

Kanerva, J. (1997). *The Java FAQ*. Reading: Addison Wesley.

KSI. (2003). *Knowledge Systems Institute Online Lecture: Network Communications*. Available from: <http://distancelearning.ksi.edu/demo/507/Ch012x.html> [7 February 2007].

Laurie, B., and Laurie, P. (1999). *Apache: The Definitive Guide*, 2<sup>nd</sup> ed. Sebastopol: O'Reilly & Associates.

Li, L., Shang, Y., and Zhang, W. (2002). Improvement of HITS-based Algorithms on Web Documents. In *the 11th international Conference on World Wide Web*, Honolulu, 7-11 May 2002.

Lie, H. W. (2006). Mobile Web Applications. In *XTech 2006: "Building Web 2.0"*, Amsterdam, 16-19 May 2006.

Linos, P. K., Ososanya, E. T., and Natarajan, H. (2001). Maintenance Support for Web Sites: A Case Study. In *the 3rd International Workshop on Web Site Evolution (WSE'01)*, Florence, 10 November 2001.

Lucent. (2000). *Characterizing End-to-End Performance: A VitalSoft Whitepaper*. Technical White Paper: Lucent Technologies NetworkCare.

Magnusson, P. S., and Montelius, J. (1997). *Performance Debugging and Tuning using an Instruction-Set Simulator*. Report No. T97-02: Swedish Institute of Computer Science.

Martin, J., and Martin, L. (2001). Web Site Maintenance with Software-Engineering Tools. In *the 3rd International Workshop on Web Site Evolution (WSE'01)*, Florence, 10 November 2001.

Matera, M., Maurino, A., Ceri, S., and Fraternali, P. (2003). Model-Driven Design of Collaborative Web Applications. *Software - Practice and Experience*, 33: 1-32.

Mogul, J. C., Douglass, F., Feldmann, A., and Krishnamurthy, B. (1997). Potential Benefits Of Delta Encoding and Data Compression for HTTP. In *the 13th ACM SIGCOMM'97*, Cannes, 14-18 September 1997.

Nagappan, R., Skoczylas, R., and Sriganesh, R. P. (2003). *Developing Java Web Services: Architecting and Developing Secure Web Services Using Java*. Indianapolis: Wiley Publishing.

Nahum, E. M. (2002). Deconstructing SPECweb99. In *the 7th International Workshop on Web Content Caching and Distribution (WCW)*, Boulder, 14-16 August 2002.

Nakano, T., Harumoto, K., Shimojo, S., and Nishio, S. (2002). User Adaptive Content Delivery Mechanism on the World Wide Web. In *the 2002 ACM Symposium on Applied Computing*, Madrid, 11-14 March 2002.

Olshefski, D., and Nieh, J. (2006). Understanding the Management of Client Perceived Response Time. In *Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS/Performance 2006*, Saint-Malo, 26-30 June 2006.

Olshefski, D., Nieh, J., and Agrawal, D. (2004). Using Certes to Infer Client Response Time at the Web Server. *ACM Transactions on Computer Systems (TOCS)*, 22(1): 49-93.

Olshefski, D. P., Nieh, J., and Nahum, E. M. (2004). ksniffer: Determining the Remote Client Perceived Response Time from Live Packet Streams. In *the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, 6-8 Dec 2004.

Pekilis, B. R., and Seviara, R. E. (2003). Automatic Response Performance Monitoring for Real-time Software with Nondeterministic Behaviors. *Performance Evaluation*, 53: 1-21.

Pressman, R. S., and Ince, D. (2000). *Software Engineering: A Practitioner's Approach (European Adaptation)*, 5<sup>th</sup> ed. Maidenhead: McGraw-Hill Publishing Company.

Ramaswamy, L., Iyengar, A., Liu, L., and Douglass, F. (2004). Automatic Detection of Fragments in Dynamically Generated Web Pages. In *the 13th International Conference on World Wide Web*, New York, 17-20 May 2004.

Reese, G. (1997). *Database Programming with JDBC and Java*. Sebastopol: O'Reilly & Associates.

Renaud, K. (2004). Quantifying the Quality of Web Authentication Mechanisms: A Usability Perspective. *Journal of Web Engineering*, 3(2): 95-123.



Renaud, K., Kotz é P., and Dyk, T. v. (2001). A Mechanism for Evaluating Feedback of E-Commerce Sites. In *the 1st IFIP conference on E-Commerce, E-Business, E-Government*, Zurich, 3-5 October 2001.

Ricca, F., and Tonella, P. (2000). Web Site Analysis: Structure and Evolution. In *International Conference on Software Maintenance (ICSM'00)*, San Jose, 11-14 October 2000.

Rossi, G., Schwabe, D., and Lyardet, F. (1999). Improving Web Information Systems with Navigational Patterns. In *the 8th International Conference on World Wide Web*, Toronto, 11-14 May 1999.

Sciascio, E. D., Donini, F. M., Mongiello, M., and Piscitelli, G. (2003). Web Applications Design and Maintenance Using Symbolic Model Checking. In *the 7th European Conference on Software Maintenance and Reengineering (CSMR'03)*, Benevento, 26-28 March 2003.

Shi, W., Wright, R., Collins, E., and Karamcheti, V. (2002). Workload Characterization of a Personalized Web Site and Its Implications for Dynamic Content Caching. In *the 7th International Workshop on Web Caching and Content Distribution (WCW'02)*, Boulder, 14-16 August 2002.

Shirazi, J. (2003). *Java Performance Tuning*, 2<sup>nd</sup> ed. Sebastopol: O'Reilly & Associates.

Simser, D. A., and Sevia, R. E. (1996). Supervision of Real-Time Software Systems Using Optimistic Path Prediction and Rollbacks. In *the 7th International Symposium on Software Reliability Engineering (ISSRE '96)*, Washington, 30 October-2 November 1996.

Stallings, W. (2004). *Data and Computer Communications*. Upper Saddle River: Prentice Hall.

Strohmaier, E. (2006). Performance Complexity: An Execution Time Metric to Characterize the Transparency and Complexity of Performance. *CTWatch Quarterly*, 2(4B).

Suchman, L. A. (1987). *Plans and Situated Actions : The Problem of Human-Machine Communication*. Cambridge: Cambridge University Press.

Vardi, M. Y. (1982). The Complexity of Relational Query Languages. In *the 14th Annual ACM Symposium on Theory of Computing*, San Francisco, 5-7 May 1982.

Yarden, S. (1997). Evaluating the Performances of Electronic Commerce Systems. In *the 1997 Winter Simulation Conference*, Atlanta, 7-10 December 1997.

## **APPENDICES**

# APPENDIX 1

## NOTATIONS OF TIMES

Table A1 Notations of Times Used in this Thesis

Notation	Description
$T_d$	Rendering time, the time elapsed from the browser starts displaying a Web page until the page is displayed in its entirety.
$T_g$	Generation time, the cumulative time for the server to process all the user requests for downloading a Web page.
$T_{ir}$	Inter-request-idle-time, the time between successive requests when no previous request is being processed or served.
$T_p$	Processing time, the time for the server to process user request for a particular HTML page or its embedded object.
$T_r$	Response time, the time elapsed between the user request for a Web page is sent until the page is displayed in its entirety.
$T_s$	Serve time, the time elapsed between an HTML page or its embedded object is sent until the corresponding server response is received.
$T_{se}$	Effective serve time, the portion of perceived serve time ( $T_{sp}$ ) that is actually used for the server to process user requests and for the user requests and server responses to be transmitted over the network.
$T_{sp}$	Perceived serve time, the time elapsed from the first byte of user request is sent until the last byte of the corresponding server response is received.
$T_{se}:T_{sp}$	The ratio that indicates the effectiveness of transmission over the network.
$T_t$	Transmission time, the cumulative time to transmit all the user requests and server responses corresponding to a Web page over the network.
$T_t:T_g$	The ratio that indicates relative contribution of transmission time ( $T_t$ ) and generation time ( $T_g$ ) to the overall response time.

## APPENDIX 2

### PARSING OF HTTP REQUESTS AND RESPONSES BY THE MUFFIN PROXY SERVER

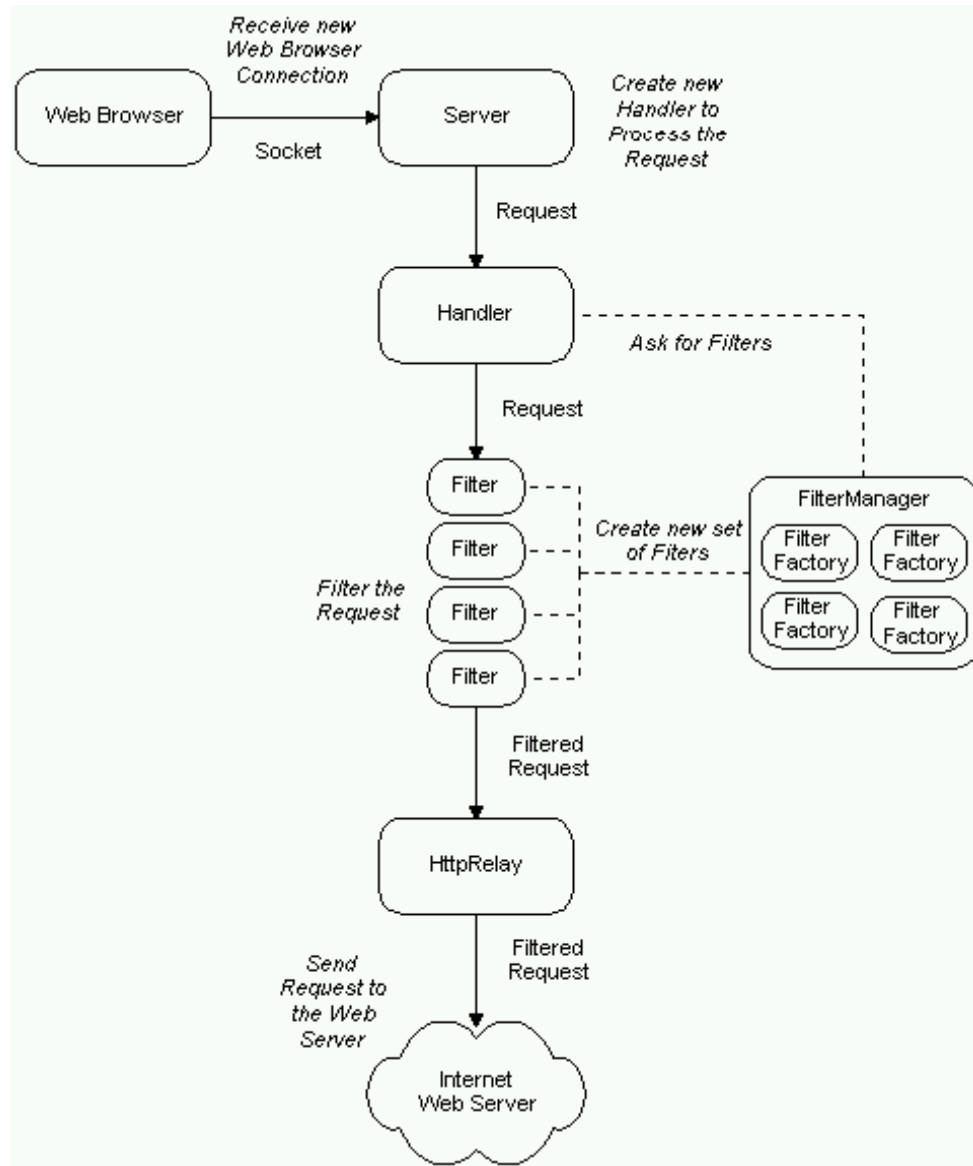


Figure A2.1 Parsing of HTTP Requests by Muffin Proxy Server

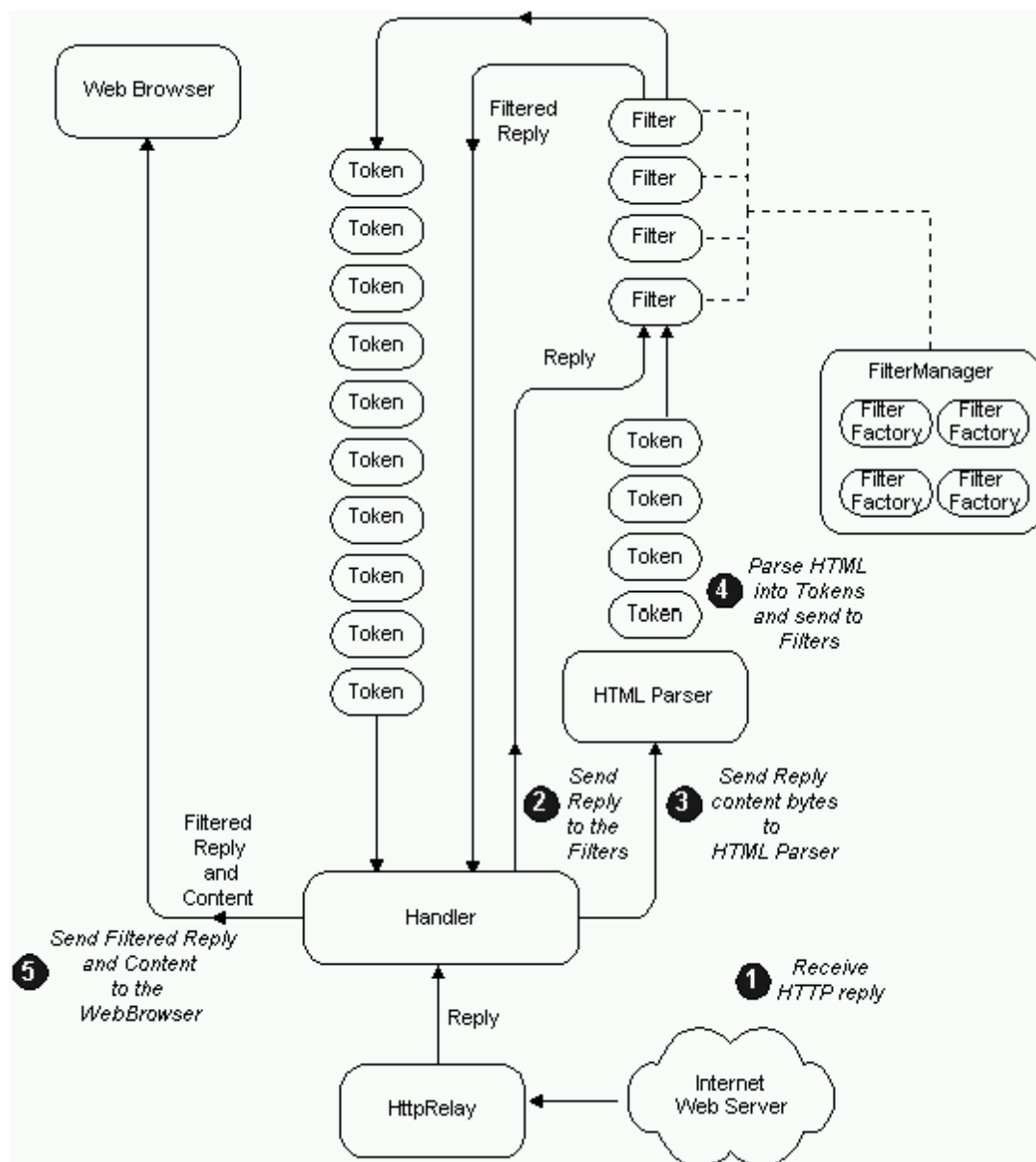


Figure A2.2 Parsing of HTTP Responses by Muffin Proxy Server

## APPENDIX 3

### RESPONSE TIME MEASUREMENT FILTER FOR MUFFIN PROXY SERVER: THE SOURCE CODE

```
* $Id: TestFilter.java,v 1.5 2000/01/24 04:02:22 boyns Exp $ */

/*
 * Copyright (C) 1996-2000 Mark R. Boyns <boyns@doit.org>
 * Amended: Thiam Kian Chiew & Marek Bell 2005
 *
 * This file is part of Muffin.
 *
 * Muffin is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * Muffin is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Muffin; see the file COPYING. If not, write to the
 * Free Software Foundation, Inc.,
 * 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
 */
package org.doit.muffin.filter;

import org.doit.muffin.*;
import org.doit.io.*;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.ByteArrayInputStream;
import java.util.Hashtable;
import java.util.Enumeration;
import java.io.FileWriter;
import java.io.IOException;

public class TestFilter implements RequestFilter, ReplyFilter {
    Prefs prefs;
    Test factory;
    Reply reply;
    Request request;

    long startTime = 0;
    long receiveTime = 0;
    private String url;

    public TestFilter(Test factory) {
        this.factory = factory;
    }
}
```

```

    public void setPrefs(Prefs prefs) {
        this.prefs = prefs;
    }

    public void filter(Request r) throws FilterException {
        startTime = System.currentTimeMillis();
        url = r.getURL();
    }

    public void filter(Reply r) throws FilterException {
        receiveTime = System.currentTimeMillis();
        System.out.println(url + "\nStart: " + startTime + " Received: " + receiveTime);
        System.out.println("\nServed in " + (receiveTime - startTime) + " ms\n");
    }

    public void close() {
    }
}

```



## APPENDIX 4

### RESPONSE TIME AND RESPONSIVENESS MONITOR: THE SOURCE CODE

```
package RTMonitor;
/**
 *runMonitor.java
 *Allows the user to select the server access log to be analysed
 *and passes the log to the analyzer to generate the monitoring report
 */
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;

public class runMonitor {

    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JDialog.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("Response Time Monitor");
        frame.getContentPane().setLayout(new FlowLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button = new JButton("Select Server Access Log");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                JFileChooser fileChooser = new JFileChooser();
                int returnValue = fileChooser.showOpenDialog(null);
                if (returnValue == JFileChooser.APPROVE_OPTION) {
                    File selectedFile = fileChooser.getSelectedFile();
                    LogIO aLogIO = new LogIO();
                    try {
                        aLogIO.parseLog(selectedFile.getPath());
                        aLogIO.logToDB();
                        VisualiseLog vl = new VisualiseLog();
                        vl.retrievePageInfo();
                        vl.createTable();
                    }
                    catch (IOException ioe){}
                }
            }
        });
        frame.getContentPane().add(button);
        frame.pack();
        frame.setVisible(true);
    }
}
```

```

package RTMonitor;
/*
 * logIO.java
 * Reads data from log file for further analysis
 */

import java.io.*;
import java.text.*;
import java.util.*;
import java.sql.*;

public class LogIO {

    private static final int LOG_FIELD_COUNT = 10;
    private ExtractLogInfo extractor;
    private ArrayList listOfPages;

    public LogIO() {
        listOfPages = new ArrayList();
    }

    private int getPageIndex(String n) {
        Iterator it = listOfPages.iterator();
        int index = -1;
        while (it.hasNext()) {
            Page pg = (Page) it.next();
            index++;
            if (pg.getName().equals(n)) {
                return index;
            }
        }
        index = -1;
        return index;
    }

    /* Parses the log file */
    public void parseLog(String logName) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader(logName));
        String line;
        extractor = new ExtractLogInfo();
        line = in.readLine();
        StringTokenizer st = new StringTokenizer(line);
        if (st.countTokens() != LOG_FIELD_COUNT) {
            System.out.println("Incompatible log file format: " + st.countTokens());
            System.exit(0);
        }
        while (line != null) {
            extractor.extractLine(line);
            line = in.readLine();
        }
    }

    public void logToDB() throws IOException{
        Connection dbconn;
        Statement stmt;
        String sql;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        }
    }
}

```

```

        catch (ClassNotFoundException cnfe) {
            cnfe.printStackTrace();
        }

    try {
        dbconn = DriverManager.getConnection("jdbc:oracle:oci8:@ugdb", "tkchiew", "xxxx");
        stmt = dbconn.createStatement();
        //clear the table
        sql = "TRUNCATE TABLE tblRequestInfo";
        stmt.executeUpdate(sql);

        Iterator iter = extractor.getListOfRequests();
        Request req = new Request();
        Calendar cld = Calendar.getInstance();
        while (iter.hasNext()) {
            sql = "";
            req = (Request)iter.next();
            if (req.getNumberOfObjects() > 0) {
                sql += "INSERT INTO tblRequestInfo (page, client, reqDate, responseTime, ";
                sql += "responsiveness, noOfObjects, responseSize) ";
                sql += "VALUES(" + req.getRequestedPage() + ", " + req.getRemoteHost() + ", ";
                cld.setTime(req.getRequestingDateAndTime());
                int month = cld.get(Calendar.MONTH)+1;
                sql += "to_date(" + cld.get(Calendar.YEAR) + "-" + month + "-";
                sql += cld.get(Calendar.DAY_OF_MONTH) + " " + cld.get(Calendar.HOUR_OF_DAY)
                    + ":";
                sql += cld.get(Calendar.MINUTE) + ":" + cld.get(Calendar.SECOND) + ", ";
                sql += "yyyy/mm/dd:hh24:mi:ss'), ";
                sql += req.getResponseTime() + ", " + req.getResponsiveness() + ", ";
                sql += req.getNumberOfObjects() + ", " + req.getResponseSize() + ")";
                stmt.executeUpdate(sql);
            }
        }
        stmt.close();
        dbconn.close();
    }
    catch (SQLException sqle) {
        sqle.printStackTrace();
    }
}

```

```

package RTMonitor;
/*
 * ExtractLogInfo.java
 * Extracts information from a log file with the format:
 * Remote host; date and time of request; first line of request;
 * status code; response size in bytes;
 */

import java.io.*;
import java.text.*;
import java.util.*;
import java.lang.*;

public class ExtractLogInfo {
    //the max time interval to be considered a request session = 60s
    public static final int SEQUENCE_RANGE = 60000; //in ms
    private String lastHost = null;
    private ArrayList listOfRequests = new ArrayList();

    /** Creates a new instance of ExtractProxy */
    public ExtractLogInfo() {}

    //Convert date and time string into an instance of Date
    //in the format of "dd/MMM/yyyy:HH:mm:ss"
    private Date getDateAndTime(String dateString) { //throws ParseException{
        SimpleDateFormat dateTimeFormat = new SimpleDateFormat("dd/MMM/yyyy:HH:mm:ss");
        Date reqDateTime = null;
        ParsePosition pp = new ParsePosition(0);
        try {
            dateString = dateString.substring(1);
            reqDateTime = dateTimeFormat.parse(dateString, pp);
        }
        catch (NullPointerException pe) {
            System.out.println("Incompatible Date Format");
        }
        return reqDateTime;
    }

    //Extract the requested object's name from the given string
    private String getObjectNames(String s) {
        StringTokenizer st;
        String t = "";
        //split the string based on '/'
        st = new StringTokenizer(s, "/");
        //get the last token, i.e. the object's name
        while (st.hasMoreTokens()) {
            t = st.nextToken();
        }
        return t;
    }

    //Determine if it is a request for a page or base object
    private boolean isAPage(String s) {
        String [] extensions = {".htm", ".html", ".shtm", ".dhtm", ".php", ".asp", ".aspx",
            ".jsp", ".cgi", ".xml", ".cfm", ".cfml", ".pl", ""};
        //StringTokenizer st;
        for (int i = 0; i < extensions.length; i++) {
            if (s.endsWith(extensions[i])) {
                return true;
            }
        }
    }
}

```

```

    }
}
return false;
}

//Extract the line read into corresponding components
public void extractLine(String line) {
    StringTokenizer st;
    Request aRequest = new Request();
    Date dateAndTime;
    String remoteHost;
    String requestedPage;
    String dateString;
    String method;
    String protocol;
    String tmp;
    int status = 0;
    long bytes = 0;
    //Split based on white space.
    //The fields are remote host; date and time of request; first line of request;
    //status code; response size in bytes;
    st = new StringTokenizer(line);
    remoteHost = st.nextToken();
    //rfc931(remote user logname) and authuser
    tmp = st.nextToken();
    tmp = st.nextToken();
    dateString = st.nextToken();
    dateAndTime = getDateAndTime(dateString);
    //Discard time locale
    st.nextToken();
    //Method: Get, Post etc
    method = st.nextToken();
    //Discard leading "
    method = method.substring(1,method.length());
    requestedPage = st.nextToken();
    //HTTP/1.0, HTTP/1.1 etc
    protocol = st.nextToken();
    //Discard trailing "
    protocol = protocol.substring(0,protocol.length()-1);
    //Get status code and bytes
    try {
        tmp = st.nextToken();
        if (!(tmp.equals("-"))) {
            status = Integer.parseInt(tmp);
        }
        tmp = st.nextToken();
        if (!(tmp.equals("-"))) {
            bytes = Long.parseLong(tmp);
        }
    }
    catch (NumberFormatException nfe) { }

    //Discard ? and trailing string for dynamic pages
    String[] s = requestedPage.split("\\?");
    //Record if a page or base object is requested
    if (isAPage(s[0])) {
        lastHost = remoteHost;
        aRequest.setRemoteHost(remoteHost);
        aRequest.setRequestedPage(s[0]);
    }
}

```

```

        aRequest.setRequestingDateAndTime(dateAndTime);
        aRequest.setResponseSize(bytes);
        listOfRequests.add(aRequest);
    }
    //Assign the non-page request to the last matching host/user if possible
    else {
        //There is previous request from the host/user
        if (!listOfRequests.isEmpty()) {
            int size = listOfRequests.size();
            boolean found = false;
            for (int i = size-1; i > -1; i--){
                aRequest = (Request) listOfRequests.get(i);
                if (aRequest.getRemoteHost().equals(remoteHost)) {
                    found = true;
                    break;
                }
            }
            //Found the matching host
            if (found) {
                //Within the session range
                if ((Math.abs(dateAndTime.getTime() -
                    aRequest.getRequestingDateAndTime().getTime())) < SEQUENCE_RANGE) {
                    ReqObject temp = new ReqObject();
                    temp.setName(s[0]);
                    temp.setSize(bytes);
                    temp.setTime(dateAndTime);
                    if (aRequest.isFirstEmbeddedObject()){
                        aRequest.setSecondAccessTime(dateAndTime);
                        aRequest.setFirstEmbeddedObject(false);
                    }
                    aRequest.setResponsiveness((int)Math.abs(aRequest.getSecondAccessTime().getTime() -
                        aRequest.getRequestingDateAndTime().getTime())/1000);
                    aRequest.setLastAccessTime(dateAndTime);
                    aRequest.setResponseTime((int)Math.abs(aRequest.getLastAccessTime().getTime() -
                        aRequest.getRequestingDateAndTime().getTime())/1000);
                    aRequest.addObjects(temp);
                }
            }
        }
        //Discard the request if no previous host/user matched
    }

    public Iterator getListOfRequests() {
        return listOfRequests.iterator();
    }
}

```

```

package RTMonitor;
/*
 * Page.java
 * Information of a page accessed by the clients
 */

import java.util.*;
import java.lang.Math;

public class Page {

    private String name;
    private int requestCount;
    private ArrayList listOfClients;
    private double averageResponseTime;
    private double medianResponseTime;
    private double responseTime_90; //90th percentile response time
    private double averageResponsiveness;
    private double medianResponsiveness;
    private double responsiveness_90; //90th percentile responsiveness
    private double medianNoOfObjects;
    private int minNoOfObjects;
    private int maxNoOfObjects;
    private double medianResponseSize;
    private long minResponseSize;
    private long maxResponseSize;

    /** Creates a new instance of Page */
    public Page() {
        name = "";
        requestCount = 0;
        listOfClients = new ArrayList();
    }

    public Page(Page pg) {
        name = pg.getName();
        requestCount = pg.getRequestCount();
        listOfClients = pg.getClients();
    }

    public String getName() {
        return name;
    }

    public int getRequestCount() {
        return requestCount;
    }

    public Iterator getListOfClients() {
        return listOfClients.iterator();
    }

    public ArrayList getClients() {
        return listOfClients;
    }

    public double getAverageResponseTime() {
        return averageResponseTime;
    }
}

```

```

public double getAverageResponsiveness() {
    return averageResponsiveness;
}

public double getMedianResponseTime() {
    return medianResponseTime;
}

public double getMedianResponsiveness() {
    return medianResponsiveness;
}

public double getMedianNoOfObjects() {
    return medianNoOfObjects;
}

public double getResponseTime_90() {
    return responseTime_90;
}

public double getResponsiveness_90() {
    return responsiveness_90;
}

public int getMinNoOfObjects() {
    return minNoOfObjects;
}

public int getMaxNoOfObjects() {
    return maxNoOfObjects;
}

public double getMedianResponseSize() {
    return medianResponseSize;
}

public long getMinResponseSize() {
    return minResponseSize;
}

public long getMaxResponseSize() {
    return maxResponseSize;
}

public void setName(String n) {
    name = n;
}

public void setRequestCount(int rc) {
    requestCount = rc;
}

public void setAverageResponseTime(double avg) {
    averageResponseTime = avg;
}

public void setAverageResponsiveness(double avg) {
    averageResponsiveness = avg;
}

```



```

public void setMedianResponseTime(double med) {
    medianResponseTime = med;
}

public void setMedianResponsiveness(double med) {
    medianResponsiveness = med;
}

public void setMedianNoOfObjects(double med) {
    medianNoOfObjects = med;
}

public void setResponseTime_90(double nin) {
    responseTime_90 = nin;
}

public void setResponsiveness_90(double nin) {
    responsiveness_90 = nin;
}

public void setMinNoOfObjects(int min) {
    minNoOfObjects = min;
}

public void setMaxNoOfObjects(int max) {
    maxNoOfObjects = max;
}

public void setMedianResponseSize(double rs) {
    medianResponseSize = rs;
}

public void setMinResponseSize(long rs) {
    minResponseSize = rs;
}

public void setMaxResponseSize(long rs) {
    maxResponseSize = rs;
}

public void addClient(Client c) {
    listOfClients.add(c);
}

public void setValues(String p, String c, Date ts, int rt, int rv, int noo, long rs) {
    name = p;
    Client cl = new Client(c, ts, rt, rv, noo, rs);
    addClient(cl);
    requestCount++;
}

public void addClient(String c, Date ts, int rt, int rv, int noo, long rs) {
    Client cl = new Client(c, ts, rt, rv, noo, rs);
    addClient(cl);
    requestCount++;
}

private double calculateAverage(int[] intArray){
    int size = intArray.length;

```

```

    int sum = 0;
    for (int i=0; i<size; i++) {
        sum += intArray[i];
    }
    return (double)sum/(double) size;
}

private void insertionSort(int[] intArray) {
    int size = intArray.length;
    int i, j, index;
    for (i=1; i<size; i++) {
        index = intArray[i];
        j = i;
        while (j>0 && intArray[j-1]>index) {
            intArray[j] = intArray[j-1];
            j--;
        }
        intArray[j] = index;
    }
}

private void insertionSort(long[] longArray) {
    int size = longArray.length;
    int i, j;
    long index;
    for (i=1; i<size; i++) {
        index = longArray[i];
        j = i;
        while (j>0 && longArray[j-1]>index) {
            longArray[j] = longArray[j-1];
            j--;
        }
        longArray[j] = index;
    }
}

public void analyse() {
    int count = listOfClients.size();
    int[] tempRT = new int[count];
    int[] tempRV = new int[count];
    int[] tempNOO = new int[count];
    long[] tempRS = new long[count];
    int i = 0, temp;
    double nin;
    Client cl;
    Iterator it = listOfClients.iterator();
    while (it.hasNext()) {
        cl = (Client)it.next();
        tempRT[i] = cl.getResponseTime();
        tempRV[i] = cl.getResponsiveness();
        tempNOO[i] = cl.getNoOfObjects();
        tempRS[i] = cl.getResponseSize();
        i++;
    }
    if (i > 0) { //there are requests recorded
        setAverageResponseTime(calculateAverage(tempRT));
        setAverageResponsiveness(calculateAverage(tempRV));
        insertionSort(tempRT);
        insertionSort(tempRV);
    }
}

```

```

insertionSort(tempNOO);
insertionSort(tempRS);
temp = i / 2;
if (i % 2 == 0) { //even number of array elements
    setMedianResponseTime(((double)tempRT[temp] + (double)tempRT[temp-1])/2.0);
    setMedianResponsiveness(((double)tempRV[temp] + (double)tempRV[temp-1])/2.0);
    setMedianNoOfObjects(((double)tempNOO[temp] + (double)tempNOO[temp-1])/2.0);
    setMedianResponseSize(((double)tempRS[temp] + (double)tempRS[temp-1])/2.0);
}
else { //odd number of array elements
    setMedianResponseTime((double)tempRT[temp]);
    setMedianResponsiveness((double)tempRV[temp]);
    setMedianNoOfObjects((double)tempNOO[temp]);
    setMedianResponseSize((double)tempRS[temp]);
}
nin = ((double) i) * 0.9;
if (Math rint(nin) == nin) { // nin is a whole number
    temp = (int) nin;
    setResponseTime_90((double)tempRT[temp]);
    setResponsiveness_90((double)tempRV[temp]);
}
else {
    temp = (int) Math.floor(nin);
    if (temp+1 >= i) {
        setResponseTime_90((double)tempRT[temp]);
        setResponsiveness_90((double)tempRV[temp]);
    }
    else {
        setResponseTime_90(((double)tempRT[temp] + (double)tempRT[temp+1])/2.0);
        setResponsiveness_90(((double)tempRV[temp] + (double)tempRV[temp+1])/2.0);
    }
}
setMinNoOfObjects(tempNOO[0]);
setMaxNoOfObjects(tempNOO[tempNOO.length-1]);
setMinResponseSize(tempRS[0]);
setMaxResponseSize(tempRS[tempNOO.length-1]);
}
}
}

```

```

package RTMonitor;
/*
 * request.java
 * Stores information about a page request.
 */

import java.util.*;
import java.io.*;

public class Request {

    private String remoteHost; //requesting host (name or IP)
    private String requestedPage; //requested page
    private ArrayList objects; //embedded objects of the page
    private Date requestingDateAndTime; //requesting date and time for the page
    private long responseSize; //response size for the page in bytes
    private Date secondAccessTime; //date and time when the first embedded
        //object was requested
    private Date lastAccessTime; //date and time when the last object was requested
    private boolean firstEmbeddedObject = true; //is there any embedded object found earlier
    private int responseTime;
    private int responsiveness;

    /** Creates a new instance of request */
    public Request() {
        remoteHost = null;
        requestedPage = null;
        objects = new ArrayList();
        requestingDateAndTime = null;
        responseSize = 0;
        secondAccessTime = null;
        lastAccessTime = null;
    }

    public void setRemoteHost(String rh) {
        remoteHost = rh;
    }

    public void setRequestedPage(String rp) {
        requestedPage = rp;
    }

    public void addObjects(ReqObject ob) {
        responseSize += ob.getSize();
        objects.add(ob);
    }

    public void setRequestingDateAndTime(Date rdt) {
        requestingDateAndTime = rdt;
    }

    public void setSecondAccessTime(Date sa) {
        secondAccessTime = sa;
    }

    public void setResponseSize(long rs) {
        responseSize = rs;
    }
}

```

```

public void setLastAccessTime(Date la) {
    lastAccessTime = la;
}

public void setFirstEmbeddedObject(boolean feo) {
    firstEmbeddedObject = feo;
}

public boolean isFirstEmbeddedObject() {
    return firstEmbeddedObject;
}

public void setResponseTime(int rt) {
    responseTime = rt;
}

public void setResponsiveness(int rv) {
    responsiveness = rv;
}

public String getRemoteHost() {
    return remoteHost;
}

public String getRequestedPage() {
    return requestedPage;
}

//return list of objects
public Iterator getObjects() {
    return objects.iterator();
}

//return object at index i
public ReqObject getObjects(int index) {
    return (ReqObject)objects.get(index);
}

public Date getRequestingDateAndTime() {
    return requestingDateAndTime;
}

public Date getSecondAccessTime() {
    return secondAccessTime;
}

public long getResponseSize() {
    return responseSize;
}

public Date getLastAccessTime() {
    return lastAccessTime;
}

public int getNumberOfObjects() {
    return objects.size();
}

public long getTotalSize() {

```

```

        long size = responseSize;
        Iterator it = objects.iterator();
        while (it.hasNext()) {
            ReqObject temp = (ReqObject) it.next();
            size += temp.getSize();
        }
        return size;
    }

    public int getResponseTime() {
        return responseTime;
    }

    public int getResponsiveness() {
        return responsiveness;
    }

    //(first embedded object request time - page request time)
    public long getPerceivedResponsiveness() {
        double time;
        try {
            if (secondAccessTime != null) {
                return Math.abs(secondAccessTime.getTime()
                    - requestingDateAndTime.getTime());
            }
            else {
                return 0;
            }
        } catch (NullPointerException npe) {
            return -1;
        }
    }

    //(last embedded object request time - page request time)
    public long getPerceivedResponseTime() {
        double time;
        try {
            if (lastAccessTime != null) {
                return Math.abs(lastAccessTime.getTime()
                    - requestingDateAndTime.getTime());
            }
            else {
                return 0;
            }
        } catch (NullPointerException npe) {
            return -1;
        }
    }
}

```

```

package RTMonitor;
/*
 * ReqObject.java
 * Represents a non-page object requested by the user.
 * With object name and response size in bytes
 */
import java.util.*;

public class ReqObject{

    private String name;
    private long size;
    private Date time;

    /** Creates a new instance of object */
    public ReqObject() {
        name = null;
        size = 0;
        time = null;
    }

    public void setName(String n) {
        name = n;
    }

    public void setSize(long s) {
        size = s;
    }

    public void setTime(Date t) {
        time = t;
    }

    public String getName(){
        return name;
    }

    public long getSize() {
        return size;
    }

    public Date getTime() {
        return time;
    }
}

```

```

package RTMonitor;
/*
 * Client.java
 * Information about clients requesting for a page
 */

import java.util.*;
import java.io.*;

public class Client {

    private String name;
    private Date timeStamp;
    private int responseTime;
    private int responsiveness;
    private int noOfObjects;
    private long responseSize;

    /** Creates a new instance of Client */
    public Client() {
        name = "";
        timeStamp = null;
        responseTime = 0;
        responsiveness = 0;
        noOfObjects = 0;
        responseSize = 0;
    }

    public Client(String n, Date ts, int rt, int rv, int noo, long rs) {
        name = n;
        timeStamp = ts;
        responseTime = rt;
        responsiveness = rv;
        noOfObjects = noo;
        responseSize = rs;
    }

    public String getName() {
        return name;
    }

    public Date getTimeStamp() {
        return timeStamp;
    }

    public int getResponseTime() {
        return responseTime;
    }

    public int getResponsiveness() {
        return responsiveness;
    }

    public int getNoOfObjects() {
        return noOfObjects;
    }

    public long getResponseSize() {
        return responseSize;
    }
}

```



```
}

public void setName(String n) {
    name = n;
}

public void setTimeStamp(Date ts) {
    timeStamp = ts;
}

public void setResponseTime(int rt) {
    responseTime = rt;
}

public void setResponsiveness(int rv) {
    responsiveness = rv;
}

public void setNoOfObjects(int noo) {
    noOfObjects = noo;
}

public void setResponseSize(long rs) {
    responseSize = rs;
}
}
```

```

package RTMonitor;
/*
 * VisualiseLog.java
 * Presents log analysis results to the users
 */
import javax.swing.*;
import java.awt.*;
import java.sql.*;
import java.text.*;
import java.util.*;
import java.io.*;
import java.lang.*;

public class VisualiseLog {
    private ArrayList listOfPages = new ArrayList();
    private java.util.Date fromDate;
    private java.util.Date toDate;

    public VisualiseLog() {
    }

    private int getPageIndex(String n) {
        Iterator it = listOfPages.iterator();
        int index = -1;
        while (it.hasNext()) {
            Page pg = (Page) it.next();
            index++;
            if (pg.getName().equals(n)) {
                return index;
            }
        }
        index = -1;
        return index;
    }

    //return the page at the given index
    public Page getPage(int index) {
        if (index > -1 && index < listOfPages.size()) {
            return (Page) listOfPages.get(index);
        }
        else {
            return null;
        }
    }

    //create table view of the page requesting information
    public void createTable() {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("Page Requests Summary");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        TableCreator newContentPane = new TableCreator(listOfPages, fromDate, toDate);
        newContentPane.setOpaque(true);
        frame.setContentPane(newContentPane);
        frame.pack();
        frame.setVisible(true);
    }

    /** Retrieve page request information from the database*/
    public void retrievePageInfo() {

```

```

        Connection dbconn;
        Statement stmt;
        ResultSet result;
        String sql;
        Page pg;
        int counter, index;

        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        }
        catch (ClassNotFoundException cnfe) {
            cnfe.printStackTrace();
        }

        try {
            dbconn = DriverManager.getConnection("jdbc:oracle:oci8:@ugdb", "tkchiew", "xxxx");
            stmt = dbconn.createStatement();
            sql = "SELECT * from tblRequestInfo ORDER BY page ASC";
            result = stmt.executeQuery(sql);

            while (result.next()) {
                index = getPageIndex(result.getString("page"));
                if (index == -1) { //A page was first requested
                    pg = new Page();
                    pg.setValues(result.getString("page"), result.getString("client"),
                                result.getTimestamp("reqDate"), result.getInt("responseTime"),
                                result.getInt("responsiveness"), result.getInt("noOfObjects"),
                                result.getLong("responseSize"));
                    listOfPages.add(pg);
                }
                else {
                    pg = (Page) listOfPages.get(index);
                    pg.addClient(result.getString("client"), result.getTimestamp("reqDate"),
                                result.getInt("responseTime"), result.getInt("responsiveness"),
                                result.getInt("noOfObjects"), result.getLong("responseSize"));
                    listOfPages.set(index, pg);
                }
            }

            sql = "SELECT Min(reqDate) as MinDate from tblRequestInfo";
            result = stmt.executeQuery(sql);
            if (result.next()) {
                fromDate = result.getTimestamp("MinDate");
            }

            sql = "SELECT Max(reqDate) as MaxDate from tblRequestInfo";
            result = stmt.executeQuery(sql);
            if (result.next()) {
                toDate = result.getTimestamp("MaxDate");
            }

            //Analyse page access information
            counter = listOfPages.size();
            index = 0;
            while (index < counter) {
                pg = (Page) listOfPages.get(index);
                pg.analyse();
                listOfPages.set(index, pg);
                index++;
            }
        }
        catch (SQLException sqle) {

```

```
        sqle.printStackTrace();
    }
}
```

```

package RTMonitor;
/**
 * MyTableModel.java
 * Build a JTable data model
 */
import java.util.*;
import java.text.*;
import javax.swing.table.*;

public class MyTableModel extends AbstractTableModel {
    private int rowCount;
    private String[] columnNames = {"Page", "No of Request", "Avg Response Time",
                                     "Avg Responsiveness"};
    private Object [][] data;

    public MyTableModel(ArrayList listOfPages) {
        rowCount = listOfPages.size();
        data = new Object[rowCount][getColumnCount()];
        for (int i = 0; i < rowCount; i++) {
            Page pg = (Page)listOfPages.get(i);
            data[i][0] = pg.getName();
            data[i][1] = new Integer(pg.getRequestCount());
            data[i][2] = new Double(pg.getAverageResponseTime());
            data[i][3] = new Double(pg.getAverageResponsiveness());
        }
    }

    public int getRowCount() {
        return rowCount;
    }

    public int getColumnCount() {
        return columnNames.length;
    }

    public Object getValueAt(int row, int col) {
        return data[row][col];
    }

    public String getColumnName(int col) {
        return columnNames[col];
    }

    public boolean isCellEditable(int row, int col) {
        return false;
    }

    public void setValueAt(Object value, int row, int col) {
        System.out.println(value.toString());
        data[row][col] = value;
        fireTableCellUpdated(row, col);
    }

    public MyTableModel getModel() {
        return this;
    }

    public Class getColumnClass(int col) {
        return getValueAt(0,col).getClass();
    }

```

}

```

package RTMonitor;
/**
 * TableCreator.java
 * Create a JTable based on the custome defined data model MyTableModel
 */

import java.util.*;
import java.text.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.event.*;
import java.awt.*;

public class TableCreator extends JPanel {
    private JTable table;
    private ArrayList LOP;
    private Date from, to;

    public TableCreator(ArrayList listOfPages, Date fromDate, Date toDate) {
        super(new GridLayout(1,0));
        LOP = listOfPages;
        from = fromDate;
        to = toDate;
        TableSorter sorter = new TableSorter(new MyTableModel(LOP));
        table = new JTable(sorter);
        sorter.setTableHeader(table.getTableHeader());
        TableColumnModel colModel = table.getColumnModel();
        colModel.getColumn(0).setPreferredWidth(350);
        colModel.getColumn(1).setPreferredWidth(150);
        colModel.getColumn(2).setPreferredWidth(150);
        colModel.getColumn(3).setPreferredWidth(150);
        DefaultTableCellRenderer tcrCol = new DefaultTableCellRenderer();
        tcrCol.setHorizontalAlignment(SwingConstants.RIGHT);
        table.setPreferredScrollableViewportSize(new Dimension(800, 300));
        table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        ListSelectionModel rowSM = table.getSelectionModel();
        ValueChangeListener vcl = new ValueChangeListener();
        rowSM.addListSelectionListener(vcl);
        //Create the scroll pane and add the table to it.
        JScrollPane scrollPane = new JScrollPane(table);
        //Add the scroll pane to this panel.
        add(scrollPane);
    }

    private class ValueChangeListener implements ListSelectionListener {
        //if a row is clicked, generate detailed report for the page shown in the row
        public void valueChanged(ListSelectionEvent e) {
            if (!e.getValueIsAdjusting()) {
                ListSelectionModel lsm = (ListSelectionModel) e.getSource();
                int row = lsm.getMinSelectionIndex();
                if (row > -1 && row < table.getRowCount()) {
                    if (table.getValueAt(row, 0) != null) {
                        Iterator it = LOP.iterator();
                        while (it.hasNext()) {
                            Page pg = (Page) it.next();
                            if (pg.getName().equals(table.getValueAt(row, 0))) {
                                ChartCreator reportChart = new ChartCreator("");
                                reportChart.createChart(pg, from, to);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

}  
}  
}  
}  
}  
}  
}  
}



```

package RTMonitor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.util.List;

import javax.swing.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.*;

/**
 * TableSorter is a decorator for TableModels; adding sorting
 * functionality to a supplied TableModel. TableSorter does
 * not store or copy the data in its TableModel; instead it maintains
 * a map from the row indexes of the view to the row indexes of the
 * model. As requests are made of the sorter (like getValueAt(row, col))
 * they are passed to the underlying model after the row numbers
 * have been translated via the internal mapping array. This way,
 * the TableSorter appears to hold another copy of the table
 * with the rows in a different order.
 * <p>
 * TableSorter registers itself as a listener to the underlying model,
 * just as the JTable itself would. Events recieved from the model
 * are examined, sometimes manipulated (typically widened), and then
 * passed on to the TableSorter's listeners (typically the JTable).
 * If a change to the model has invalidated the order of TableSorter's
 * rows, a note of this is made and the sorter will resort the
 * rows the next time a value is requested.
 * <p>
 * When the tableHeader property is set, either by using the
 * setTableHeader() method or the two argument constructor, the
 * table header may be used as a complete UI for TableSorter.
 * The default renderer of the tableHeader is decorated with a renderer
 * that indicates the sorting status of each column. In addition,
 * a mouse listener is installed with the following behavior:
 * <ul>
 * <li>
 * Mouse-click: Clears the sorting status of all other columns
 * and advances the sorting status of that column through three
 * values: {NOT_SORTED, ASCENDING, DESCENDING} (then back to
 * NOT_SORTED again).
 * <li>
 * SHIFT-mouse-click: Clears the sorting status of all other columns
 * and cycles the sorting status of the column through the same
 * three values, in the opposite order: {NOT_SORTED, DESCENDING, ASCENDING}.
 * <li>
 * CONTROL-mouse-click and CONTROL-SHIFT-mouse-click: as above except
 * that the changes to the column do not cancel the statuses of columns
 * that are already sorting - giving a way to initiate a compound
 * sort.
 * </ul>
 * <p>
 * This is a long overdue rewrite of a class of the same name that
 * first appeared in the swing table demos in 1997.
 *
 * @author Philip Milne
 * @author Brendon McLean

```

```

* @author Dan van Enkevort
* @author Parwinder Sekhon
* @version 2.0 02/27/04
*/

```

```

public class TableSorter extends AbstractTableModel {
    protected TableModel tableModel;

    public static final int DESCENDING = -1;
    public static final int NOT_SORTED = 0;
    public static final int ASCENDING = 1;
    private static Directive EMPTY_DIRECTIVE = new Directive(-1, NOT_SORTED);
    public static final Comparator COMPARABLE_COMAPRATOR = new Comparator() {
        public int compare(Object o1, Object o2) {
            return ((Comparable) o1).compareTo(o2);
        }
    };
    public static final Comparator LEXICAL_COMPARATOR = new Comparator() {
        public int compare(Object o1, Object o2) {
            return o1.toString().compareTo(o2.toString());
        }
    };

    private Row[] viewToModel;
    private int[] modelToView;
    private JTableHeader tableHeader;
    private MouseListener mouseListener;
    private TableModelListener tableModelListener;
    private Map columnComparators = new HashMap();
    private List sortingColumns = new ArrayList();

    public TableSorter() {
        this.mouseListener = new MouseHandler();
        this.tableModelListener = new TableModelHandler();
    }

    public TableSorter(TableModel tableModel) {
        this();
        setTableModel(tableModel);
    }

    public TableSorter(TableModel tableModel, JTableHeader tableHeader) {
        this();
        setTableHeader(tableHeader);
        setTableModel(tableModel);
    }

    private void clearSortingState() {
        viewToModel = null;
        modelToView = null;
    }

    public TableModel getTableModel() {
        return tableModel;
    }

    public void setTableModel(TableModel tableModel) {
        if (this.tableModel != null) {
            this.tableModel.removeTableModelListener(tableModelListener);

```

```

    }

    this.tableModel = tableModel;
    if (this.tableModel != null) {
        this.tableModel.addTableModelListener(tableModelListener);
    }

    clearSortingState();
    fireTableStructureChanged();
}

public JTableHeader getTableHeader() {
    return tableHeader;
}

public void setTableHeader(JTableHeader tableHeader) {
    if (this.tableHeader != null) {
        this.tableHeader.removeMouseListener(mouseListener);
        TableCellRenderer defaultRenderer = this.tableHeader.getDefaultRenderer();
        if (defaultRenderer instanceof SortableHeaderRenderer) {
            this.tableHeader.setDefaultRenderer(((SortableHeaderRenderer)
defaultRenderer).tableCellRenderer);
        }
    }
    this.tableHeader = tableHeader;
    if (this.tableHeader != null) {
        this.tableHeader.addMouseListener(mouseListener);
        this.tableHeader.setDefaultRenderer(
            new SortableHeaderRenderer(this.tableHeader.getDefaultRenderer()));
    }
}

public boolean isSorting() {
    return sortingColumns.size() != 0;
}

private Directive getDirective(int column) {
    for (int i = 0; i < sortingColumns.size(); i++) {
        Directive directive = (Directive)sortingColumns.get(i);
        if (directive.column == column) {
            return directive;
        }
    }
    return EMPTY_DIRECTIVE;
}

public int getSortingStatus(int column) {
    return getDirective(column).direction;
}

private void sortingStatusChanged() {
    clearSortingState();
    fireTableDataChanged();
    if (tableHeader != null) {
        tableHeader.repaint();
    }
}

public void setSortingStatus(int column, int status) {

```

```

        Directive directive = getDirective(column);
        if (directive != EMPTY_DIRECTIVE) {
            sortingColumns.remove(directive);
        }
        if (status != NOT_SORTED) {
            sortingColumns.add(new Directive(column, status));
        }
        sortingStatusChanged();
    }

    protected Icon getHeaderRendererIcon(int column, int size) {
        Directive directive = getDirective(column);
        if (directive == EMPTY_DIRECTIVE) {
            return null;
        }
        return new Arrow(directive.direction == DESCENDING, size,
            sortingColumns.indexOf(directive));
    }

    private void cancelSorting() {
        sortingColumns.clear();
        sortingStatusChanged();
    }

    public void setColumnComparator(Class type, Comparator comparator) {
        if (comparator == null) {
            columnComparators.remove(type);
        } else {
            columnComparators.put(type, comparator);
        }
    }

    protected Comparator getComparator(int column) {
        Class columnType = tableModel.getColumnClass(column);
        Comparator comparator = (Comparator) columnComparators.get(columnType);
        if (comparator != null) {
            return comparator;
        }
        if (Comparable.class.isAssignableFrom(columnType)) {
            return COMPARABLE_COMAPRATOR;
        }
        return LEXICAL_COMPARATOR;
    }

    private Row[] getViewToModel() {
        if (viewToModel == null) {
            int tableModelRowCount = tableModel.getRowCount();
            viewToModel = new Row[tableModelRowCount];
            for (int row = 0; row < tableModelRowCount; row++) {
                viewToModel[row] = new Row(row);
            }

            if (isSorting()) {
                Arrays.sort(viewToModel);
            }
        }
        return viewToModel;
    }
}

```

```

public int modelIndex(int viewIndex) {
    return getViewToModel()[viewIndex].modelIndex;
}

private int[] getModelToView() {
    if (modelToView == null) {
        int n = getViewToModel().length;
        modelToView = new int[n];
        for (int i = 0; i < n; i++) {
            modelToView[modelIndex(i)] = i;
        }
    }
    return modelToView;
}

// TableModel interface methods

public int getRowCount() {
    return (tableModel == null) ? 0 : tableModel.getRowCount();
}

public int getColumnCount() {
    return (tableModel == null) ? 0 : tableModel.getColumnCount();
}

public String getColumnName(int column) {
    return tableModel.getColumnName(column);
}

public Class getColumnClass(int column) {
    return tableModel.getColumnClass(column);
}

public boolean isCellEditable(int row, int column) {
    return tableModel.isCellEditable(modelIndex(row), column);
}

public Object getValueAt(int row, int column) {
    return tableModel.getValueAt(modelIndex(row), column);
}

public void setValueAt(Object aValue, int row, int column) {
    tableModel.setValueAt(aValue, modelIndex(row), column);
}

// Helper classes

private class Row implements Comparable {
    private int modelIndex;

    public Row(int index) {
        this.modelIndex = index;
    }

    public int compareTo(Object o) {
        int row1 = modelIndex;
        int row2 = ((Row) o).modelIndex;

        for (Iterator it = sortingColumns.iterator(); it.hasNext();) {

```

```

        Directive directive = (Directive) it.next();
        int column = directive.column;
        Object o1 = tableModel.getValueAt(row1, column);
        Object o2 = tableModel.getValueAt(row2, column);

        int comparison = 0;
        // Define null less than everything, except null.
        if (o1 == null && o2 == null) {
            comparison = 0;
        } else if (o1 == null) {
            comparison = -1;
        } else if (o2 == null) {
            comparison = 1;
        } else {
            comparison = getComparator(column).compare(o1, o2);
        }
        if (comparison != 0) {
            return directive.direction == DESCENDING ? -comparison : comparison;
        }
    }
    return 0;
}

private class TableModelHandler implements TableModelListener {
    public void tableChanged(TableModelEvent e) {
        // If we're not sorting by anything, just pass the event along.
        if (!isSorting()) {
            clearSortingState();
            fireTableChanged(e);
            return;
        }

        // If the table structure has changed, cancel the sorting; the
        // sorting columns may have been either moved or deleted from
        // the model.
        if (e.getFirstRow() == TableModelEvent.HEADER_ROW) {
            cancelSorting();
            fireTableChanged(e);
            return;
        }

        // We can map a cell event through to the view without widening
        // when the following conditions apply:
        //
        // a) all the changes are on one row (e.getFirstRow() == e.getLastRow()) and,
        // b) all the changes are in one column (column != TableModelEvent.ALL_COLUMNS) and,
        // c) we are not sorting on that column (getSortingStatus(column) == NOT_SORTED) and,
        // d) a reverse lookup will not trigger a sort (modelToView != null)
        //
        // Note: INSERT and DELETE events fail this test as they have column == ALL_COLUMNS.
        //
        // The last check, for (modelToView != null) is to see if modelToView
        // is already allocated. If we don't do this check; sorting can become
        // a performance bottleneck for applications where cells
        // change rapidly in different parts of the table. If cells
        // change alternately in the sorting column and then outside of
        // it this class can end up re-sorting on alternate cell updates -
        // which can be a performance problem for large tables. The last

```

```

        // clause avoids this problem.
        int column = e.getColumn();
        if (e.getFirstRow() == e.getLastRow()
            && column != TableModelEvent.ALL_COLUMNS
            && getSortingStatus(column) == NOT_SORTED
            && modelToView != null) {
            int viewIndex = getModelToView()[e.getFirstRow()];
            fireTableChanged(new TableModelEvent(TableSorter.this,
                viewIndex, viewIndex,
                column, e.getType()));

            return;
        }

        // Something has happened to the data that may have invalidated the row order.
        clearSortingState();
        fireTableDataChanged();
        return;
    }
}

private class MouseHandler extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {
        JTableHeader h = (JTableHeader) e.getSource();
        TableColumnModel columnModel = h.getColumnModel();
        int viewColumn = columnModel.getColumnIndexAtX(e.getX());
        int column = columnModel.getColumn(viewColumn).getModelIndex();
        if (column != -1) {
            int status = getSortingStatus(column);
            if (!e.isControlDown()) {
                cancelSorting();
            }
            // Cycle the sorting states through {NOT_SORTED, ASCENDING, DESCENDING} or
            // {NOT_SORTED, DESCENDING, ASCENDING} depending on whether shift is pressed.
            status = status + (e.isShiftDown() ? -1 : 1);
            status = (status + 4) % 3 - 1; // signed mod, returning {-1, 0, 1}
            setSortingStatus(column, status);
        }
    }
}

private static class Arrow implements Icon {
    private boolean descending;
    private int size;
    private int priority;

    public Arrow(boolean descending, int size, int priority) {
        this.descending = descending;
        this.size = size;
        this.priority = priority;
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        Color color = c == null ? Color.GRAY : c.getBackground();
        // In a compound sort, make each successive triangle 20%
        // smaller than the previous one.
        int dx = (int)(size/2*Math.pow(0.8, priority));
        int dy = descending ? dx : -dx;
        // Align icon (roughly) with font baseline.
        y = y + 5*size/6 + (descending ? -dy : 0);
    }
}

```

```

int shift = descending ? 1 : -1;
g.translate(x, y);

// Right diagonal.
g.setColor(color.darker());
g.drawLine(dx / 2, dy, 0, 0);
g.drawLine(dx / 2, dy + shift, 0, shift);

// Left diagonal.
g.setColor(color.brighter());
g.drawLine(dx / 2, dy, dx, 0);
g.drawLine(dx / 2, dy + shift, dx, shift);

// Horizontal line.
if (descending) {
    g.setColor(color.darker().darker());
} else {
    g.setColor(color.brighter().brighter());
}
g.drawLine(dx, 0, 0, 0);

g.setColor(color);
g.translate(-x, -y);
}

public int getIconWidth() {
    return size;
}

public int getIconHeight() {
    return size;
}
}

private class SortableHeaderRenderer implements TableCellRenderer {
    private TableCellRenderer tableCellRenderer;

    public SortableHeaderRenderer(TableCellRenderer tableCellRenderer) {
        this.tableCellRenderer = tableCellRenderer;
    }

    public Component getTableCellRendererComponent(JTable table,
                                                    Object value,
                                                    boolean isSelected,
                                                    boolean hasFocus,
                                                    int row,
                                                    int column) {
        Component c = tableCellRenderer.getTableCellRendererComponent(table,
            value, isSelected, hasFocus, row, column);
        if (c instanceof JLabel) {
            JLabel l = (JLabel) c;
            l.setHorizontalTextPosition(JLabel.LEFT);
            int modelColumn = table.convertColumnIndexToModel(column);
            l.setIcon(getHeaderRendererIcon(modelColumn, l.getFont().getSize()));
        }
        return c;
    }
}

```



```
private static class Directive {  
    private int column;  
    private int direction;  
  
    public Directive(int column, int direction) {  
        this.column = column;  
        this.direction = direction;  
    }  
}  
}
```

```

package RTMonitor;
/**
 * ChartCreator.java
 * Creates detailed report.
 * Uses JFreeChart, a free chart library for the Java(tm) platform
 * (C) Copyright 2000-2007, by Object Refinery Limited and Contributors.
 */

import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.GradientPaint;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.lang.Integer;
import java.util.Iterator;
import java.util.Date;
import java.text.SimpleDateFormat;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JLabel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.axis.CategoryLabelPositions;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.BarRenderer;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class ChartCreator extends ApplicationFrame {

    public ChartCreator(String title) {
        super(title);
    }

    //creates the dataset
    private static CategoryDataset createDataset(int[] values) {
        // row keys
        String series1 = "Number of Users";

        // column keys
        String category1 = "< 3";
        String category2 = "3 - 8";
        String category3 = "8 - 15";
        String category4 = "> 15";

        // create the dataset
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.addValue(values[0], series1, category1);

```

```

        dataset.addValue(values[1], series1, category2);
        dataset.addValue(values[2], series1, category3);
        dataset.addValue(values[3], series1, category4);
        return dataset;
    }

    //Creates a bar chart to show response time distribution
    private static JFreeChart createResponseTimeChart(CategoryDataset dataset) {
        // create the chart
        JFreeChart chart = ChartFactory.createBarChart(
            "Response Time Distribution",    // chart title
            "Time (s)",                    // domain axis label
            "Count",                        // range axis label
            dataset,                        // data
            PlotOrientation.VERTICAL,      // orientation
            true,                           // include legend
            true,                           // tooltips?
            false                           // URLs?
        );
        // set the background color for the chart
        chart.setBackgroundPaint(Color.white);
        // get a reference to the plot for further customisation...
        CategoryPlot plot = (CategoryPlot) chart.getPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setDomainGridlinePaint(Color.white);
        plot.setDomainGridlinesVisible(true);
        plot.setRangeGridlinePaint(Color.white);
        final NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
        // disable bar outlines
        BarRenderer renderer = (BarRenderer) plot.getRenderer();
        renderer.setDrawBarOutline(false);
        // set up gradient paints for series...
        GradientPaint gp2 = new GradientPaint(0.0f, 0.0f, Color.red,
            0.0f, 0.0f, new Color(64, 0, 0));
        renderer.setSeriesPaint(0, gp2);
        CategoryAxis domainAxis = plot.getDomainAxis();
        domainAxis.setCategoryLabelPositions(
            CategoryLabelPositions.createUpRotationLabelPositions(
                Math.PI / 6.0));
        return chart;
    }

    //Creates a bar chart to show responsiveness distribution
    private static JFreeChart createResponsivenessChart(CategoryDataset dataset) {
        // create the chart
        JFreeChart chart = ChartFactory.createBarChart(
            "Responsiveness Distribution",    // chart title
            "Time (s)",                    // domain axis label
            "Count",                        // range axis label
            dataset,                        // data
            PlotOrientation.VERTICAL,      // orientation
            true,                           // include legend
            true,                           // tooltips?
            false                           // URLs?
        );
        // set the background color for the chart
        chart.setBackgroundPaint(Color.white);
    }

```

```

// get a reference to the plot for further customisation
CategoryPlot plot = (CategoryPlot) chart.getPlot();
plot.setBackgroundPaint(Color.lightGray);
plot.setDomainGridlinePaint(Color.white);
plot.setDomainGridlinesVisible(true);
plot.setRangeGridlinePaint(Color.white);
final NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
// disable bar outlines
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setDrawBarOutline(false);
// set up gradient paints for series
GradientPaint gp0 = new GradientPaint(0.0f, 0.0f, Color.blue,
    0.0f, 0.0f, new Color(0, 0, 64));
renderer.setSeriesPaint(0, gp0);
CategoryAxis domainAxis = plot.getDomainAxis();
domainAxis.setCategoryLabelPositions(
    CategoryLabelPositions.createUpRotationLabelPositions(
        Math.PI / 6.0));
return chart;
}

//group response times into 4 categories, i.e. < 3; 3-8; 8-15; >15
public void countResponseTime(Page pg, int[] values) {
    Iterator it = pg.getListOfClients();
    while (it.hasNext()) {
        Client cl = (Client) it.next();
        if (cl.getResponseTime() < 3) {
            values[0]++;
        }
        else if (cl.getResponseTime() >= 3 && cl.getResponseTime() < 8) {
            values[1]++;
        }
        if (cl.getResponseTime() >= 8 && cl.getResponseTime() < 15) {
            values[2]++;
        }
        if (cl.getResponseTime() >= 15) {
            values[3]++;
        }
    }
}

//group responsiveness into 4 categories, i.e. < 3; 3-8; 8-15; >15
public void countResponsiveness(Page pg, int[] values) {
    Iterator it = pg.getListOfClients();
    while (it.hasNext()) {
        Client cl = (Client) it.next();
        if (cl.getResponsiveness() < 3) {
            values[0]++;
        }
        else if (cl.getResponsiveness() >= 3 && cl.getResponsiveness() < 8) {
            values[1]++;
        }
        if (cl.getResponsiveness() >= 8 && cl.getResponsiveness() < 15) {
            values[2]++;
        }
        if (cl.getResponsiveness() >= 15) {
            values[3]++;
        }
    }
}

```

```

    }
}

//creates a bar chart for the given page monitored within the specified period
public void createChart(Page pg, Date from, Date to) {
    ReportFrame frame = new ReportFrame(pg, from, to);
    frame.setTitle("Detailed Report: " + pg.getName());
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frame.pack();
    frame.show();
}

class ReportFrame extends JFrame {
    //set a frame as the container
    public ReportFrame(Page pg, Date from, Date to) {
        final int FRAME_WIDTH = 650;
        final int FRAME_HEIGHT = 700;
        setSize(FRAME_WIDTH, FRAME_HEIGHT);
        int[] counts = {0, 0, 0, 0};
        String d1, d2;
        SimpleDateFormat formatter = new SimpleDateFormat("dd MMM yyyy HH:mm:ss");
        d1 = formatter.format(from);
        d2 = formatter.format(to);

        //set page information
        JLabel nameLabel = new JLabel(" Page name: " + pg.getName());
        JLabel periodLabel = new JLabel(" Period: " + d1 + " to " + d2);
        JLabel noOfRequestsLabel = new JLabel(" Number of Requests: " +
            pg.getRequestCount());
        JLabel medNoOfObjectsLabel = new JLabel(" Number of Embedded Objects (Median): " +
            pg.getMedianNoOfObjects());
        JLabel minNoOfObjectsLabel = new JLabel(" Number of Embedded Objects (Min): " +
            pg.getMinNoOfObjects());
        JLabel maxNoOfObjectsLabel = new JLabel(" Number of Embedded Objects (Max): " +
            pg.getMaxNoOfObjects());
        JLabel medResponseSizeLabel = new JLabel(" Response Size in Bytes (Median): " +
            pg.getMedianResponseSize());
        JLabel minResponseSizeLabel = new JLabel(" Response Size in Bytes (Min): " +
            pg.getMinResponseSize());
        JLabel maxResponseSizeLabel = new JLabel(" Response Size in Bytes (Max): " +
            pg.getMaxResponseSize());
        JLabel avgResponseTimeLabel = new JLabel(" Response Time (Mean): " +
            pg.getAverageResponseTime());
        JLabel medResponseTimeLabel = new JLabel(" Response Time (Median): " +
            pg.getMedianResponseTime());
        JLabel ninResponseTimeLabel = new JLabel(" Response Time (90th-Percentile): " +
            pg.getResponseTime_90());
        JLabel avgResponsivenessLabel = new JLabel(" Responsiveness (Mean): " +
            pg.getAverageResponsiveness());
        JLabel medResponsivenessLabel = new JLabel(" Responsiveness (Median): " +
            pg.getMedianResponsiveness());
        JLabel ninResponsivenessLabel = new JLabel(" Responsiveness (90th-Percentile): " +
            pg.getResponseTime_90());
        JPanel pageInfoPanel = new JPanel();
        pageInfoPanel.setLayout(new GridLayout(16, 1));
        pageInfoPanel.add(nameLabel);
        pageInfoPanel.add(periodLabel);
        pageInfoPanel.add(noOfRequestsLabel);

```

```

        pageInfoPanel.add(medNoOfObjectsLabel);
        pageInfoPanel.add(minNoOfObjectsLabel);
        pageInfoPanel.add(maxNoOfObjectsLabel);
        pageInfoPanel.add(medResponseSizeLabel);
        pageInfoPanel.add(minResponseSizeLabel);
        pageInfoPanel.add(maxResponseSizeLabel);
        pageInfoPanel.add(avgResponseTimeLabel);
        pageInfoPanel.add(medResponseTimeLabel);
        pageInfoPanel.add(ninResponseTimeLabel);
        pageInfoPanel.add(avgResponsivenessLabel);
        pageInfoPanel.add(medResponsivenessLabel);
        pageInfoPanel.add(ninResponsivenessLabel);

        countResponseTime(pg, counts);
        CategoryDataset dataset = createDataset(counts);
        JFreeChart chart = createResponseTimeChart(dataset);
        ChartPanel chartPanelRT = new ChartPanel(chart, false);
        chartPanelRT.setPreferredSize(new Dimension(300, 200));
        JScrollPane spane1 = new JScrollPane(chartPanelRT);

        for (int i=0; i<4; i++) {
            counts[i] = 0;
        }
        countResponsiveness(pg, counts);
        dataset = createDataset(counts);
        chart = createResponsivenessChart(dataset);
        ChartPanel chartPanelRV = new ChartPanel(chart, false);
        chartPanelRV.setPreferredSize(new Dimension(300, 200));
        JScrollPane spane2 = new JScrollPane(chartPanelRV);

        Container contentPane = getContentPane();
        contentPane.add(pageInfoPanel, "North");
        contentPane.add(spane1, "Center");
        contentPane.add(spane2, "South");
    }
}
}

```

## APPENDIX 5

### DATABASE QUERIES EXAMINING TOOL: THE SOURCE CODE

```
package pageMaintenance;
/**
 *startMaintenance.java
 *Allows the user to select the Web page to be analysed
 */
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;

public class startMaintenance {

    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JDialog.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("Database Queries Analyser");
        frame.getContentPane().setLayout(new FlowLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button = new JButton("Select Web Page");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                JFileChooser fileChooser = new JFileChooser();
                int returnValue = fileChooser.showOpenDialog(null);
                if (returnValue == JFileChooser.APPROVE_OPTION) {
                    File selectedFile = fileChooser.getSelectedFile();
                    dbQueryAnalyser dqa = new dbQueryAnalyser();

                    try {
                        dqa.getQuery(selectedFile.getPath());
                    }
                    catch (IOException ioe){}
                }
            }
        });
        frame.getContentPane().add(button);
        frame.pack();
        frame.setVisible(true);
    }
}
```

```

package pageMaintenance;
import java.io.*;
import java.util.*;

/*
 * dbQueryAnalyser.java
 * Examines database queries in a .cfm file
 */

public class dbQueryAnalyser {
    private ArrayList listOfQueries;
    private boolean hasQuery;

    public dbQueryAnalyser() {
        hasQuery = false;
        listOfQueries = new ArrayList();
    }

    //remove comments from in and write the results to temp
    public void removeComments(BufferedReader in, BufferedWriter temp) throws IOException {
        String line = in.readLine();
        boolean commentFound = false;
        int index1, index2, beginIndex;
        while (line != null) {
            line = line.trim();
            beginIndex = 0;
            boolean repeat = true;
            while (repeat == true) {
                if (commentFound == true) { //comment spans from previous line
                    index2 = line.indexOf("-->");
                    if (index2 != -1) { //closing comment tag found
                        commentFound = false;
                        if ((index2 + 3) == line.length()) { //no more string after the closing comment tag
                            temp.write("\n");
                            repeat = false;
                            break;
                        }
                    }
                    else { //more strings after the closing comment tag
                        beginIndex = index2 + 3;
                        line = line.substring(beginIndex, line.length()); //search remaining strings
                    }
                }
                else { //closing comment tag not found
                    temp.write("\n");
                    repeat = false;
                    break;
                }
            }
            index1 = line.indexOf("<!--");
            if (index1 != -1) { //comment found
                commentFound = true;
                index2 = line.indexOf("-->", index1+1);
                if (index2 != -1) { //closing comment tag found
                    if (index1 > 0) {
                        temp.write(line.substring(beginIndex, index1-1)); //copy parts before comment
                    }
                    commentFound = false;
                    if ((index2 + 3) == line.length()) { //no more string after closing comment tag
                        temp.write("\n");
                    }
                }
            }
            else {
                temp.write(line);
            }
            line = in.readLine();
        }
    }
}

```



```

        repeat = false;
        break;
    }
    else { //more strings after closing comment tag
        beginIndex = index2 + 3;
        line = line.substring(beginIndex, line.length()); //search remaining strings
    }
}
else { //closing comment tag not found
    temp.write("\n");
    repeat = false;
    break;
}
}
else { //no comment found in this line
    temp.write(line + "\n");
    repeat = false;
    break;
}
}
line = in.readLine();
}
}

```

```

//search in the query for the given string (find)
//mainly to identify query name and datasource
private String search(String query, String find) {
    int index1, index2;
    String queryName = null;
    index1 = query.toLowerCase().indexOf(find.toLowerCase());
    if (index1 != -1) {
        index1 = query.indexOf('=', index1+1);
        index1 = query.indexOf('"', index1+1);
        index2 = query.indexOf('"', index1+1);
        if ((index1 != -1) && (index2 != -1)) {
            queryName = query.substring(index1+1, index2).trim();
        }
    }
    return queryName;
}

```

```

//extract query name, data source, and data fields retrieved from the query
public void extractDataFields(String query) {
    int index1, index2;
    dbQuery dbq = new dbQuery();
    String queryName = "";
    String dataSource = "";
    String temp = "";
    String table = "";
    String dataField = "";
    String alias = "";
    //extract query name
    if ((queryName = search(query, " name")) == null) {
        dbq.setName("");
    }
    else {
        dbq.setName(queryName);
    }
}

```

```

//Extract datasource name
if ((dataSource = search(query, " datasource")) == null) {
    System.out.println("Invalid query. Datasource name not specified.");
    System.exit(0);
}
else {
    dbq.setDataSource(dataSource);
}

//Extract data fields and table names
index1 = query.toLowerCase().indexOf("select ");
if (index1 != -1) {
    index2 = query.toLowerCase().indexOf("from ", index1);
}
else {
    index2 = -1;
}
while ((index1 != -1) && (index2 != -1)) {
    //string between select and from
    dbq.setIsSelect(true);
    temp = query.substring(index1+7, index2).trim();
    StringTokenizer st1 = new StringTokenizer(temp, ",");
    String current;
    int index3, index4;
    //identified fields separated by comma (,)
    while (st1.hasMoreTokens()) {
        current = st1.nextToken().trim();
        alias = "";
        table = "";
        dataField = current;
        //SQL functions, like count, max, etc
        index3 = current.indexOf('(');
        if (index3 != -1) {
            index4 = current.indexOf(')', index3+1);
            if (index4 != -1) {
                dataField = current.substring(index3+1, index4).trim();
                index3 = current.indexOf(' ', index4+1); // more strings
            }
            else {
                System.out.println("Query syntax error");
                System.exit(0);
            }
        }
        //data field has alias
        else if ((index3 = current.indexOf(' ')) != -1) {
            dataField = current.substring(0, index3);
        }
        else {
            dataField = current;
        }
        //data field qualified with table name
        index4 = dataField.indexOf('.');
        if (index4 != -1) {
            table = dataField.substring(0, index4);
            dataField = dataField.substring(index4+1, dataField.length());
        }
        //extract alias
        if (index3 != -1) {
            alias = current.substring(index3+1, current.length()).trim();
        }
    }
}

```

```

        if (alias.toLowerCase().startsWith("as ")) {
            alias = alias.substring(2, alias.length()).trim();
        }
        else {
            System.out.println("Query syntax error");
            System.exit(0);
        }
    }
    //wildcard used in the query
    if (dataField.equals("") && alias.equals("")) {
        dbq.setHasAsterisk(true);
    }

    //set the data field and add to the query
    dataField field = new dataField(dataField, alias, table);
    dbq.addDataField(field);
}
//search for other select ... from statement in the query
index1 = query.toLowerCase().indexOf(" select ", index2);
index2 = query.toLowerCase().indexOf(" from ", index1);
}
//add the query to the list
listOfQueries.add(dbq);
}

//check if the given character is an alphabet, digit, or underscore
public boolean isAlphaNumericUnderscore(char c) {
    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') ||
        (c >= '0' && c <= '9') || (c == '_')) {
        return true;
    }
    else {
        return false;
    }
}

//check the occurrences of queries and data fields in the file
//line numbers for the occurrences are identified
public void checkOccurrence(File tempFile) throws IOException {
    String queryName, line, query, data;
    int index1, index2, index3, lineNumber, listIndex, dataIndex;
    Iterator it = listOfQueries.iterator();
    listIndex = 0;
    //search for each query in the list
    while (it.hasNext()) {
        BufferedReader in = new BufferedReader(new FileReader(tempFile));
        dbQuery dbq = (dbQuery) it.next();
        if (dbq.getIsSelect() == true) {
            queryName = dbq.getName();
            line = in.readLine();
            lineNumber = 1;
            while ((line != null) && (!queryName.equals(""))) {
                query = queryName.toLowerCase();
                index1 = line.toLowerCase().indexOf(query);
                //make sure the query found is not just a substring
                if ((index1 > 0) && (index1+query.length() < line.length())) {
                    if(isAlphaNumericUnderscore(line.charAt(index1-1)) ||
                        isAlphaNumericUnderscore(line.charAt(index1+query.length()))) {
                        index1 = -1;
                    }
                }
            }
        }
    }
}

```

```

    }
}
//query found
while (index1 != -1) {
    //data field specified after the query?
    index2 = line.indexOf('.', index1);
    if ((index2 != -1) && (index2 == index1 + query.length())) {
        index3 = index2 + 1;
        //possible data field?
        if ((index3 < line.length()) && (isAlphaNumericUnderscore(line.charAt(index3)))) {
            while (isAlphaNumericUnderscore(line.charAt(index3))) {
                index3++;
                if (index3 >= line.length()) {
                    break;
                }
            }
        }
        //extract possible data field
        data = line.substring(index2+1, index3);
        dataIndex = -1;
        //a data field specified in the query?
        for (int i = 0; i < dbq.getNumberOfDataFields(); i++) {
            if (((dataField)dbq.getListOfDataFields().get(i)).getName().equals(data) ||
                ((dataField)dbq.getListOfDataFields().get(i)).getAlias().equals(data)) {
                dataIndex = i;
                break;
            }
        }
        //is a data field specified in the query
        if (dataIndex != -1) {
            ((dataField)dbq.getListOfDataFields().get(dataIndex)).addOccurrence(lineNumber);
        }
        //not a data field but wildcard was specified in the query
        else if (dbq.getHasAsterisk() == true) {
            dataField df = new dataField(data, "", "");
            df.addOccurrence(lineNumber);
            ((dbQuery)listOfQueries.get(listIndex)).addDataField(df);
        }
        //search remaining of the line
        index1 = line.toLowerCase().indexOf(query, index1+1);
    }
    //not a data field after the period (.) following query name
    else {
        index1 = line.toLowerCase().indexOf(query, index1+1);
    }
}
//query name found, but not followed by a period (.)
else {
    dataIndex = -1;
    //query treated as a whole
    //found before this in the file?
    for (int i = 0; i < dbq.getNumberOfDataFields(); i++) {
        if (((dataField)dbq.getListOfDataFields().get(i)).getName().equals("-")) {
            dataIndex = i;
            break;
        }
    }
}
//first occurrence in the file, created an entry in the query
if (dataIndex == -1) {
    dataField df1 = new dataField("-", "", "");
}

```

```

        df1.addOccurrence(lineNumber);
        ((dbQuery)listOfQueries.get(listIndex)).addDataField(df1);
    }
    //occurred in earlier part(s), update the entry
    else {
        dataField df1 = ((dbQuery)listOfQueries.get(listIndex)).getDataField(dataIndex);
        df1.addOccurrence(lineNumber);
        ((dbQuery)listOfQueries.get(listIndex)).setDataField(dataIndex, df1);
    }
    //search remaining of the line
    index1 = line.toLowerCase().indexOf(query, index1+1);
}
//if query name found in remaining of the line
//make sure it is not just a substring
while ((index1 > 0) && (index1+query.length() < line.length())) {
    if(isAlphaNumericUnderscore(line.charAt(index1-1)) ||
        isAlphaNumericUnderscore(line.charAt(index1+query.length()))) {
        index1 = line.toLowerCase().indexOf(query, index1+1);
    }
    //just a substring
    else {
        break;
    }
}
}
//read the next line of the file
line = in.readLine();
lineNumber++;
}
}
//search for the next query in the list
listIndex++;
in.close();
}

}

//write query and data field occurrences to output file
public void writeOccurrence(BufferedWriter out) throws IOException {
    Iterator it = listOfQueries.iterator();
    while (it.hasNext()) {
        dbQuery dbq = (dbQuery)it.next();
        if (dbq.getIsSelect() == true) {
            out.write("Query Name: " + dbq.getName());
            out.write("\nDatasource: " + dbq.getDataSource());
            Iterator it1 = dbq.getListOfDataFields().iterator();
            while (it1.hasNext()) {
                dataField df = (dataField)it1.next();
                out.write("\nData Field: " + df.getName());
                if (!df.getAlias().equals("")) {
                    out.write("    Alias: " + df.getAlias());
                }
                if (!df.getTable().equals("")) {
                    out.write("    Table: " + df.getTable());
                }
                out.write("\nOccurs at line(s): ");
                Iterator it2 = df.getListOfOccurrence().iterator();
                while (it2.hasNext()) {
                    out.write(((Integer)it2.next()).intValue() + " ");
                }
            }
        }
    }
}

```

```

    }
  }
  out.write("\n\n");
}
}
}

```

//examine the queries, including their existence, in the given file

```

public void getQuery(String fName) throws IOException {
    File outFile = new File ("D:\\out.dat");
    FileWriter fout = new FileWriter(outFile);
    BufferedWriter out = new BufferedWriter(fout);
    File inFile = new File (fName);
    BufferedReader in = new BufferedReader(new FileReader(inFile));
    boolean found = false;
    String query = "";
    //check if it is a .cfm file
    if (fName.endsWith(".cfm") || fName.endsWith(".CFM")) {
        out.write("Page Name: " + fName + ":\n");
        out.write("Queries found:\n");
        String buffer = in.readLine();
        while (buffer != null) {
            StringTokenizer words = new StringTokenizer(buffer, " ");
            while (words.hasMoreTokens()) {
                String nextTok = words.nextToken();
                //System.out.println(nextTok);
                //<CFQUERY> or <CFSTOREPROC> found
                if (nextTok.trim().toUpperCase().startsWith("<CFQUERY") ||
                    nextTok.trim().toUpperCase().startsWith("<CFSTOREPROC")) {
                    found = true;
                    hasQuery = true;
                    out.write(nextTok + " ");
                    query = query + nextTok + " ";
                    while (words.hasMoreTokens() && found) {
                        nextTok = words.nextToken();
                        if (!(nextTok.trim().toUpperCase().equals("</CFQUERY>")) &&
                            !(nextTok.trim().toUpperCase().equals("</CFSTOREPROC>"))){
                            out.write(nextTok + " ");
                            query = query + nextTok + " ";
                        }
                    }
                    else {
                        found = false;
                        out.write(nextTok + " \n\n");
                        query = query + nextTok + "\n";
                        extractDataFields(query);
                        query = "";
                    }
                }
            }
        }
    }
    //Query spans to the next line
    else if (found){
        while (!(nextTok.trim().toUpperCase().equals("</CFQUERY>")) &&
            !(nextTok.trim().toUpperCase().equals("</CFSTOREPROC>")) &&
            found){
            out.write(nextTok + " ");
            query = query + nextTok + " ";
            if (words.hasMoreTokens()) {
                nextTok = words.nextToken();
            }
        }
    }
}

```

```

        else {
            break;
        }
    }
    if (nextTok.trim().toUpperCase().equals("</CFQUERY>") ||
        nextTok.trim().toUpperCase().equals("</CFSTOREPROC>")) {
        out.write(nextTok + " \n\n");
        query = query + nextTok + "\n";
        found = false;
        extractDataFields(query);
        query = "";
    }
}
buffer = in.readLine();
}
}
else {
    System.out.println("File type not match");
    System.exit(0);
}
//Database query found in the file
if (hasQuery) {
    inFile = new File (fName);
    in = new BufferedReader(new FileReader(inFile));
    File tempFile = new File ("D:\\temp.dat");
    FileWriter tempout = new FileWriter(tempFile);
    BufferedWriter temp = new BufferedWriter(tempout);
    //remove comments from in, return the file without comments as temp
    removeComments(in, temp);
    temp.close();
    in.close();
    //check the occurrences of data fields in temp
    checkOccurrence(tempFile);
    //write occurrences to output file
    out.write("Number of queries: " + listOfQueries.size() + "\n\n");
    writeOccurrence(out);
}
else {
    out.write("No query found:\n");
}
out.close();
}
}

```

```

package pageMaintenance;

/*
 * dbQuery.java
 * Information about query created: query name, data source and data fields retrieved
 */

import java.util.ArrayList;

public class dbQuery {
    private ArrayList listOfDataFields;
    private String name;
    private String dataSource;
    private boolean hasAsterisk;
    private boolean isSelect; //select from query?

    /** Creates a new instance of query */
    public dbQuery() {
        name = "";
        dataSource = "";
        hasAsterisk = false;
        isSelect = false;
        listOfDataFields = new ArrayList();
    }

    public String getName() {
        return name;
    }

    public void setName(String n) {
        name = n;
    }

    public String getDataSource() {
        return dataSource;
    }

    public void setDataSource(String ds) {
        dataSource = ds;
    }

    public boolean getHasAsterisk() {
        return hasAsterisk;
    }

    public void setHasAsterisk(boolean ha) {
        hasAsterisk = ha;
    }

    public boolean getIsSelect() {
        return isSelect;
    }

    public void setIsSelect(boolean is) {
        isSelect = is;
    }

    public void addDataField(dataField df) {
        listOfDataFields.add(df);
    }

```



```
}

public dataField getDataField(int i) {
    return (dataField) listOfDataFields.get(i);
}

public void setDataField(int i, dataField df) {
    listOfDataFields.set(i, df);
}

public ArrayList getListOfDataFields() {
    return listOfDataFields;
}

public int getNumberOfDataFields() {
    return listOfDataFields.size();
}
}
```

```

package pageMaintenance;
/*
 * dataField.java
 * Data field retrieved and information about its occurrence
 */

import java.lang.Integer;
import java.lang.String;
import java.util.ArrayList;

public class dataField {
    private ArrayList listOfOccurrence;
    private String name; //data field name
    private String alias; //alias given to the data field
    private String table; //table in which the data field is stored

    /** Creates a new instance of fieldName */
    public dataField(String n, String a, String t) {
        name = n;
        alias = a;
        table = t;
        listOfOccurrence = new ArrayList();
    }

    public String getName() {
        return name;
    }

    public void setName(String n) {
        name = n;
    }

    public String getAlias() {
        return alias;
    }

    public void setAlias(String a) {
        alias = a;
    }

    public String getTable() {
        return table;
    }

    public void setTable(String t) {
        table = t;
    }

    public void addOccurrence(int lineNumber) {
        listOfOccurrence.add(new Integer(lineNumber));
    }

    public ArrayList getListOfOccurrence() {
        return listOfOccurrence;
    }

    public int numberOfOccurrence() {
        return listOfOccurrence.size();
    }
}

```

}

## APPENDIX 6

### COLDFUSION FILES USED TO TEST THE DATABASE QUERIES EXAMINING TOOL

#### Page A

```
<cfsetting enablecfoutputonly="Yes">

<!-- If session has timed out, start again -->

<cfif not isdefined("session.id")>
    <cflocation url="index.cfm" addToken="No">
</cfif>

<cfif #go# eq " Everything is OK ">
<cfset session.completionMessage = "Thanks for checking your details">
<cflocation url="notices.cfm">
</cfif>

<cftry>
<cfif isdefined("isdeacon")>
<cfquery datasource="xxxxx" username="xxxxx" password="xxxx">
    update deacon
    set email='#memEmail#', role='#role#' where id = '#session.id#'
</cfquery>
</cfif>

<cfset atpos = #Find("@",#memNickname#)#>

<cfif atpos gt 0> <!-- it has an @ in it -->
    <!-- only email address may have @ in them -->
    <cfset session.completionMessage = "Sorry, please try another username, your username
should not have an @ in it">

<cfelse>
<!-- check to see if someone else already has the chosen login name -->
<cfquery name="checkLogin" datasource="xxxxx" username="xxxxx" password="xxxx">

    select count(*) as count from L4PERSON where
    nickname='#memNickname#' and id <> '#session.id#'

</cfquery>

<cfif checkLogin.count eq 0> <!-- no one is using it -->

<cfquery name="insertPerson" datasource="xxxxx" username="xxxxx" password="xxxx">

    update L4PERSON
    set email='#memEmail#', nickname='#memNickname#'
    where id = #session.id#

</cfquery>
```

```

<cfset session.completionMessage = "Personal Details changed">

<cfelse> <!-- already in use -->

<cfset session.completionMessage = "Sorry, please try another username, that one is already being
used by someone else">

</cfif>
</cfif>

<cfif session.Username neq #memEmail#>
<!-- now go through all other tables with email addresses -->

<cfquery datasource="xxxxx" username="xxxxx" password="xxxx">
update remindtea set email='#memEmail#' where email='#session.Username#'
</cfquery>

<cfquery datasource="xxxxx" username="xxxxx" password="xxxx">
update remind set email='#memEmail#' where email='#session.Username#'
</cfquery>

<cfquery datasource="xxxxx" username="xxxxx" password="xxxx">
update remindcommunity set email='#memEmail#' where email='#session.Username#'
</cfquery>

<cfquery datasource="xxxxx" username="xxxxx" password="xxxx">
update steward set email='#memEmail#' where email='#session.Username#'
</cfquery>
</cfif>

<cfoutput>onroll #onroll#<br> </cfoutput>
<cfif isDefined("onroll")>
<cfquery name="address" datasource="xxxxx" >
    select * from address where id=#addressid#
</cfquery>

<cfif #address.details# neq #details# or #address.telephone# neq #telephone#>

<!-- changed - check for shared-->

<cfif not isDefined("shared")><cfset shared="No"></cfif>

<cfif #shared# eq "yes" and #sharedopt# eq 'no'>
<cfoutput>don't change for everyone<br> </cfoutput>
<!-- need to create new address record -->
<cfquery name="insertActivity" datasource="xxxxx" username="xxxxx" password="xxxx">
    insert into address

```

```

        set details="#details#", telephone="#telephone#", postcode="#postcode#"
    </cfquery>
    <cfquery name="getnum" datasource="xxxxx" username="xxxxx" password="xxxx">
        select id from address
        where details="#details#" and telephone="#telephone#"
    </cfquery>

    <cfquery name="insertActivity2" datasource="xxxxx" username="xxxxx" password="xxxx">
        update person
        set address=#getnum.id#
        where id=#rollid#
    </cfquery>

    <cfelse> <!-- change in place -->
    <cfoutput>change for everyone<br> </cfoutput>
    <cfquery name="insertActivity" datasource="xxxxx" username="xxxxx" password="xxxx">
        update address
        set details="#details#", telephone="#telephone#"
        where id=#addressid#
    </cfquery>

    </cfif>

</cfif>

<cfset session.Username = #memEmail#>

</cfif>

<cfcatch>

    <cflocation url="logerror.cfm?message=#cfcatch.message#&action=Could not update
personal details for #memEmail#&forwardpage=main.cfm">

</cfcatch>
</cftry>

<cflocation url="notices.cfm">

```

## **Page B**

```
<cfsetting enablecfoutputonly="Yes">

<!-- Cannot skip stages! --->
<cfif not IsDefined("selected")>
    <cflocation url="index.cfm" addToken="No">
</cfif>

<!-- If session has timed out, start again --->
<cfif not isdefined("session.id")>
    <cflocation url="index.cfm" addToken="No">
</cfif>

<!-- Get the time of submission --->

<cfset session.time4 = #DateFormat(Now(),"DDDD, MMMM DD, YYYY")# >
<cfset session.time4 = session.time4 & " " & #TimeFormat(Now(),"HH,mm,ss")#>

<!-- save process info in the session --->
<cfset session.chosenDoodle = selected>

<cfif (session.pos1 eq session.chosenPIN)
    and (session.pos3 eq session.chosenPostcode)
    and (session.pos4 eq session.chosenDoodle)>
    <cfset session.loginsuccess = "YES">
    <cfset session.tries=0>
    <cfset session.postcodetries=0>
    <cfset session.doodletries=0>

<cfelse>
    <cfset session.loginsuccess = "NO">
</cfif>

<!-- Shared html --->
<cfoutput>
<html>
<head>
    <title>#Application.Title#</title>
<meta http-equiv="Expires" content="Thu, 01 Dec 1994 16:00:00 GMT" />
</cfoutput>

<cfif session.loginsuccess eq "NO">
    <cfoutput> <!-- Login Failed page --->
    </head>
    <body bgcolor="###ff9090">
    <div align="center">
    <h1>Sorry - Login Unsuccessfull</h1><br>

    <h3><a href="index.cfm">Click here to try again</a></h3>
    </div>
    </body>
    </html>
    </cfoutput>
<cfelse>

<cfoutput>

</head>
```

```

<body bgcolor="##9fff9f">
<div align="center">

<h1>Login Successful!</h1><br>

<h3>Please wait, acessing site...</h3>

</div>

</cfoutput>
</cfif>

<cftry>

<cfquery name="ins_attempt" datasource="xxxxx" username="xxxxx" password="xxxx">
    INSERT INTO l4attempt
        set id='#session.id#', time='#session.time0#', success='#session.loginsuccess#';
</cfquery>
<cfquery name="get_attempt_id" datasource="xxxxx" username="xxxxx" password="xxxx">
    select aid from l4attempt where id='#session.id#' and time='#session.time0#'
</cfquery>
<cfset attemptId=get_attempt_id.aid>

<!--1-->

<cfquery name="ins_stage1" datasource="xxxxx" username="xxxxx" password="xxxx">
    INSERT INTO l4stage
        set attempt='#attemptId#', type='1', time='#session.time0#',
        chosen='#session.chosenPIN#', location='#session.pos1#'
</cfquery>
<cfquery name="getSid" datasource="xxxxx" username="xxxxx" password="xxxx">
    select sid from l4stage where attempt='#attemptId#' and type='1' and
    time='#session.time0#' and chosen='#session.chosenPIN#'and location='#session.pos1#'
</cfquery>
<cfset sid=getSid.sid>

<cfloop index="i" from="1" to="10">
    <cfquery name="ins_item1" datasource="xxxxx" username="xxxxx" password="xxxx">
        INSERT INTO l4item
            values( '#sid#', '#i#', '#session.pins[i]#', '#session.dir1[i]#')
    </cfquery>
</cfloop>

<!--3-->

<cfquery name="ins_stage3" datasource="xxxxx" username="xxxxx" password="xxxx">
    INSERT INTO l4stage
        set attempt='#attemptId#', type='3', time='#session.time3#',
        chosen='#session.chosenPostcode#', location='#session.pos3#'
</cfquery>

<cfquery name="getSid3" datasource="xxxxx" username="xxxxx" password="xxxx">
    select sid from l4stage where attempt='#attemptId#' and type='3' and
    time='#session.time3#' and chosen='#session.chosenPostcode#'and location='#session.pos3#'
</cfquery>
<cfset sid=getSid3.sid>

<cfloop index="i" from="1" to="10">
    <cfquery name="ins_item3" datasource="xxxxx" username="xxxxx" password="xxxx">

```



```

        INSERT INTO l4item
        VALUES ('#sid#','#i#','postcode','#session.dir3[i]#')
    </cfquery>
</cfloop>

<!--4-->
<cfquery name="ins_stage4" datasource="xxxxx" username="xxxxx" password="xxxx">
    INSERT INTO l4stage
    set
    attempt='#attemptId#',type='4',time='#session.time4#',chosen='#session.chosenDoodle#',location='#
    session.pos4#'
</cfquery>
<cfquery name="getSid4" datasource="xxxxx" username="xxxxx" password="xxxx">
    select sid from l4stage where attempt='#attemptId#' and type='4' and
    time='#session.time4#' and chosen='#session.chosenDoodle#'and location='#session.pos4#'
</cfquery>
<cfset sid=getSid4.sid>

<cfloop index="i" from="1" to="12">
    <cfquery name="ins_item4" datasource="xxxxx" username="xxxxx" password="xxxx" >
        INSERT INTO l4item
        VALUES ('#sid#','#i#','doodle','#session.dir4[i]#')
    </cfquery>
</cfloop>
<cfcatch>
<!-- don't do anything here... -->
</cfcatch>
</cftry>
<cfoutput></body></html></cfoutput>

<cfif session.loginsuccess eq "YES">
    <cflocation url="notices.cfm" addToken="No">
</cfif>

<!-- /LOG INFORMATION IN THE DATABASE!!! -->

```

## Page C

```
<!-- If session has timed out, start again --->
<cfif not IsDefined("session.id")>
    <cflocation url="index.cfm" addToken="No">
</cfif>

<cfquery name="checkdeacon" datasource="xxxxx">
    select count(*) as count from deacon where id = '#session.id#'
</cfquery>

<cfset session.thispage = "news">

<cfinclude template="log.cfm">

<cfquery name="getpages" datasource="xxxxx">
    select * from pages where pagename='#session.thispage#'
</cfquery>
<cfset session.thisPageURL=#getpages.url#>
<cfset session.thisPageHeading = #getpages.pageheading#>

<cfoutput>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <meta name="keywords" content="" />
    <meta name="description" content="" />
    <title>Helensburgh Baptist Church</title>

    <script type="text/javascript"></script>

    <style type="text/css" media="screen">
        @import "zen2.css";
    </style>
</head>

<body>
<div id="container">
<cfinclude template="includelinks2.cfm">

    <div id="introgap"><br></div>

    <div id="intro">

        <div id="preamble">
            <h2>#session.thisPageHeading#</h2>
        </div>

    </div>

    <div id="introgap"><br></div>

    <div id="supportingText">
```

```

<div id="explanation">

    <h2><span>Welcome, #session.name#!
#session.adminMessage#</span></h2>

    <cfif isdefined("session.completionMessage")>
    <cfif session.completionMessage neq "">
    <table align="center" ><tr><td width=100% bgcolor="##99CCFF">
        <b><big>#session.completionMessage#</big></b>
    </td></tr></table>
    <cfset session.completionMessage = "">
    </cfif></cfif>

<!-- now get special announcements -->
<cfquery name="getann" datasource="xxxxx">
    select * from announce order by number desc
</cfquery>
<!-- put them into arrays for display -->
<cfset today = #Now()#>
<cfset todayDay = #Day(today)#>
<cfset todayMonth = #Month(today)#>
<cfset todayYear = #Year(today)#>
<cfset nextYear = todayYear+1>
<cfset today = #CREATEODBCDATE("#todayMonth#/#todayDay#/#todayYear#")#>

<cfset nums = ArrayNew(1)>
<cfset days = ArrayNew(1)>
<cfset months = ArrayNew(1)>
<cfset years = ArrayNew(1)>
<cfset announce = ArrayNew(1)>

<cfset i = 1>
<cfloop query="getann">
    <CFSET newdate = #CREATEODBCDATE("#month#/#day#/#year#")#>

    <!-- relevant announcement? -->
    <cfif #newdate# gte #today#>
        <cfset nums[i]= number>
        <cfset days[i] = day>
        <cfset months[i] = month>
        <cfset years[i] = year>
        <cfset announce[i] = activity>
        <cfset i = i + 1>
    </cfif>
</cfloop>

<!-- only display if there are current announcements -->
    

<p>
<cfif ArrayLen(days) gt 0>

<cfloop index="i" from ="1" to= "#ArrayLen(days)#">

    <div id="announce"><span>
#announce[i]#</span>
    <cfif checkdeacon.count eq 1 and session.admin eq 1>
        <table border=1><tr>

```

```

<td>Expires:
#days[i]##/months[i]##/years[i]##</td>

<td><form action="remove.cfm" method="post">
<input type="hidden" name="remove" value="#nums[i]#">
    <input type="hidden" name="table" value="announce">
    <input type="hidden" name="forwardpage" value="main.cfm">
    <input type="hidden" name="type" value="announcement">

value="Remove">
    <input src="delete-icon.gif" type="image" name="RemoveActiv"

</form></td>

<td><form action="editannounce.cfm" method="post" >
<input type="hidden" name="edit" value="#nums[i]#">
<input type="image" src="edit.gif" name="EditActiv" value="Change">
</form></td>

<td><form action="extend.cfm" method="post" >
    <input type="hidden" name="extendweek" value="#nums[i]#">
    <input src="plus.gif" type="image" name="Extend"

value="Extend">

</form></td>

method="post" >
<td><form action="mailmembers.cfm?message=#announce[i]#"

<input src="emailicon.gif" type="image" name="Eml" value="Eml">
</form></td>

<td><form action="move.cfm" method="post">

    <input type="hidden" name="extendweek" value="#nums[i]#">
    <input src="move.png" type="image" name="Move" value="Move">
</form></td>

<td><form action="moveToBreaking.cfm" method="post">

    <input type="hidden" name="mv" value="#nums[i]#">
    <input src="breaking.gif" type="image" name="MoveBreak">
</form></td>

</tr>

<tr><td>Actions:</td><td>Bin</td><td>Edit</td><td>Extend</td><td>Mail</td><td>Move to
archive</td><td>Move to Breaking News</td></tr>
</table>

</cfif>

</div>

</cfloop>

</cfif> <!-- announcements to be displayed -->

<cfinclude template="footer.cfm">
</div>

```

```

<div id="admincontainer">
<cfif checkdeacon.count eq 1 and session.admin eq 1>
<div id="admin">
<h3 class="admin">Deacons' Admin</h3>

    <h4>Add Announcement:</h4>
    <form action="addannounce.cfm" method="post">
        How many weeks should this announcement stay on the page? (Default is 1
week)<br>
        <select name="weeks">
            <cfloop index="i" from ="1" to= "4">
                <option value="#i#">#i# Week(s)</option>
            </cfloop>
        </select>
        <br>
        Short Description:          <input type="text" size=60 name="heading">
        <br>
        Announcement<br>
        <textarea name="ann" rows=10 cols=60 style="bgcolor:##FFFFCC;">
        </textarea><br>
        <input type="submit" name="go" value=" Add Announcement "
            style="background-color:green; color:white; font-weight:bold; ">
    </form>

    <p>

</div>
</cfif> <!-- deacons -->

<cfif session.name eq "Xxxx">

<div id="admin">
<h2>Xxxx's Admin</h2>
<a href="getYouthmessage.cfm" target="_blank">Send email to the youth</a>
<br><a href="admin.cfm" target="_blank">Admin</a>
<br><a href="getNewMember.cfm" target="_blank">Add New Member</a>
<br><a href="getPrayer.cfm" target="_blank">Update Prayer List</a>
    <h4>
        <form action="clearduties.cfm" method="post">
            <input type="submit" name="go" value=" Clear Old Deacon Duties ">
        </form></h4>
    <h4>
        <form action="clearact.cfm" method="post">
            <input type="submit" name="go" value=" Clear Old Activities ">
        </form></h4>
    <p>

</div>

</cfif>

</div>

</body>
</html>

</cfoutput>

```

## **Page D**

```
<!-- If session has timed out, start again ---->

<cfif not IsDefined("session.id")>
    <cflocation url="index.cfm" addToken="No">
</cfif>

<cfset today = #Now()#>
<cfset todayDay = #Day(today)#>
<cfset todayMonth = #Month(today)#>
<cfset todayYear = #Year(today)#>
<cfset nextYear = todayYear+1>
<cfset today = #CREATEODBCDATE("#todayMonth#/#todayDay#/#todayYear#")#>

<!-- get the activities -->
<!-- this is for the developments list that tells about recent
    -- developments -->
<cfquery name="getall" datasource="xxxxx">
select number, activity from building order by number desc
</cfquery>

<!-- now pack it into the arrays for display -->
<cfset sentences = ArrayNew(1)>
<cfset numbers = ArrayNew(1)>

<cfset i = 1>
<cfloop query="getall">
    <cfset sentences[i] = activity>
    <cfset numbers[i] = number>
    <cfset i = i + 1>
</cfloop>

<!-- check for building curator -->
<cfquery name="check" datasource="xxxxx">
select count(*) as count from builders where id = #session.id#
</cfquery>

<!-- week details -->
<cfquery name="weeks" datasource="xxxxx">
    select * from week
</cfquery>

<!-- Get the list of building schedule -->
<cfquery name="getsched" datasource="xxxxx">
    select * from buildsched order by startweek
</cfquery>

<!-- put all details into arrays-->
<cfset wk = ArrayNew(1)>
<cfset wkday = ArrayNew(1)>
<cfset wkmonth = ArrayNew(1)>
<cfset wkyear = ArrayNew(1)>
<cfset i = 1>
<cfloop query="weeks">
    <cfset wk[i] = number>
    <cfset wkday[i] = day>
    <cfset wkmonth[i] = month>
```

```

        <cfset wkyear[i] = year>
        <cfset i = i+1>
    </cfloop>

    <!-- get the activities - put all details into arrays-->
    <cfset buildnum = ArrayNew(1)>
    <cfset buildstart = ArrayNew(1)>
    <cfset buildend = ArrayNew(1)>
    <cfset fabricz = ArrayNew(1)>
    <cfset servicesz = ArrayNew(1)>
    <cfset paintz = ArrayNew(1)>
    <cfset carpetz = ArrayNew(1)>
    <cfset activityz = ArrayNew(1)>
    <cfset allactivz = ArrayNew(1)>
    <cfset i = 1>
    <cfloop query="getsched">
        <cfset buildnum[i] = number>
        <cfset buildstart[i] = startweek>
        <cfset buildend[i] = endweek>
        <cfset fabricz[i] = fabric>
        <cfset servicesz[i] = services>
        <cfset paintz[i] = paint>
        <cfset carpetz[i] = carpet>
        <cfset activityz[i] = activity>
        <cfset allactivz[i] = allactiv>
        <cfset i = i+1>
    </cfloop>

    <cfoutput>

    <html>
    <META name="DESCRIPTION" content="Helensburgh Baptist Church">
    <META name="KEYWORDS" content="Baptist, Helensburgh, Jesus, Christian">
    <head lang="en">
    <title>Helensburgh Baptist Church</title>
    </head>

    <!-- STYLE ===== -->
    <style>
    @import url(mem.css);
    </style>

    <!-- STYLE ===== -->
    <body class="hbc" >
    <!-- ##### -->
    <!-- ##### TOP BAR ##### -->
    
    <img <img
    <cfif #session.admin# eq 0>
    src= "pics.gif"
    <cfelse>
    src= "constructionPic.png"
    </cfif>
    alt='Members Website' align="center" >
    <br clear="left">
    <!-- ##### -->
    <!-- ##### END TOP BAR ##### -->
    <!-- ##### -->

```

```

<table width="100%" >
<tr>
<!-- ===== -->
<!-- LEFT HAND LINKS PANEL ---->
<!-- ===== -->

<td width="15%" valign="top" >

<cfquery name="getpages" datasource="xxxxx">
select * from pages
</cfquery>

<cfset thisPage = "building">

<table>
<tr><td>
        <b>Pages:</b>
</td></tr>

<!-- loop through the pages and produce them -->
<cfloop query="getpages">

<cfset showThis = 1> <!-- the next if may modify this -->

<!-- if the deacons page is being looped through, check
      whether this person is a deacon -->
<cfif #pagename# eq 'deacons'>

        <cfquery name="checkdeacon" datasource="xxxxx">
            select count(*) as count from deacon where id = '#session.id#'
        </cfquery>
        <cfif checkdeacon.count neq 1>
            <cfset showThis = 0>
        </cfif>
</cfif>

<cfif showThis eq 1>
<tr>
        <!-- if this is the page we're on it is a different colour
              and doesn't have a link -->
        <cfif #pagename# eq #thisPage#>
            <cfset thisPageHeading = #pageheading#>
            <cfset thisPageURL = #url#>
            <td bordercolor="##000000" bgcolor="##FFFF80" class="links">
                <!-- home gets an image -->
                <cfif #pagename# eq "home">
                    
                </cfif>

                #pagename#
            <cfelse>

                <td bgcolor="##FF9900">
                <cfif #pagename# eq "home"></cfif>
                <a class="plain" href="#url#">
                    #pagename#
                </a>
            </cfif>

```



```

</td>
</tr>

</cfif> <!-- showThis -->

</cfloop> <!-- query="getpages" -->
</table> <!-- page buttons -->

<!-- actions now -->
<table>
<tr><td>
<b>Actions:</b>
</td></tr>

<tr><td>
<a href="logout.cfm" class="plain">

</a></td></tr>

<tr><td>
<a href="personal.cfm" class="plain">

</a></td></tr>

<cfif checkdeacon.count eq 1>
    <tr><td>
        <cfif session.admin eq 0>
            <a href="doadmin.cfm?page=#thisPageURL#" class="plain">
                
            <cfelse>
                <a href="donotadmin.cfm?page=#thisPageURL#" class="plain">
                    
                </cfif>
            </a></td></tr>

</cfif> <!-- checkdeacon -->

<tr><td>
<a class="plain" href="mailto:zennana@onetel.com?subject=MemberContact">

</a></td></tr>
</table> <!-- actions -->

</td>

<td width="70%" valign="top">
<!-- =====>
<!-- THIS IS WHERE THE CONTENT CHANGES ---->
<!-- =====>
<h1 align="center">#thisPageHeading# #session.adminMessage#</h1>

<cfif isdefined("session.completionMessage")>
<table align="center" ><tr><td width=100% bgcolor="##99CCFF">
<b><large>#session.completionMessage#</large></b>
</td></tr></table>
<cfset session.completionMessage = "">

```

```

</cfif>

<p>
<b>

<!-- first display the announcements of developments at the
      top of the page -->
<table border=1>
<cfloop index="i" from ="1" to= "#ArrayLen(sentences)#">
<tr>
<td>#sentences[i]#</td>
<cfif check.count eq 1 and session.admin eq 1>
      <td>
        <!-- allow the ccurators to remove at will -->
        <form action="rembuilding.cfm" method="post">
          <input type="hidden" name="go" value="#numbers[i]#">
          <input src="delete-icon.gif" type="image" name="remove" value="Remove">
        </form>
      </td>
</cfif>
</tr>
</cfloop>
</table>

<p></p>

<p>

<cfquery name="check2" datasource="xxxxx">
      select count(*) as count from L4PERSON where id = '#session.id#'
      and member = 1
</cfquery>

<cfif check2.count eq 0>

<cfelse>

<!-- now generate the table containing the schedule -->
<div align="center"><h2>Work Schedule</h2></div>
<table border=1>
<tr><th colspan=2>Time</th><th colspan=4>Activity</th>
</tr>
<tr><th colspan=4 align="left">(Current week shaded)</th><th colspan=2><a
href="##finishing">Finishing Work</a></th></tr>

<tr><th>Month</th><th>Week Num</th><th><a href="##fabric">Building & Fabric</a></th>
<th><a href="##services">Services & Fittings</a></th><th><a href="##paint">Painting &
Decorating</a></th><th><a href="##carpet">Carpeting & outfitting</a></th></tr>

<cfset theweek = #Week(today)# - 18>

<cfset theMonth = 'none'>
<cfloop index="i" from ="1" to= "#ArrayLen(wk)#">

      <cfif wk[i] eq theweek>
        <cfset theColour = "##FF9999">
      <cfelse>

```

```

        <cfset theColour = "###FFFFFF">
    </cfif>

    <cfset prevMonth = theMonth>
    <cfset theMonth = wkmonth[i]>

    <tr>
    <!-- month and day -->
    <td bgcolor=#theColour#
        #wkday[i]#
    <cfif prevMonth neq theMonth>
        #MonthAsString(theMonth)#
    </cfif>
    </td>

    <!-- week number -->
    <td bgcolor=#theColour#
        #wk[i]#
    </td>

    <!-- store info for each column -->
    <cfset foundfabric = 0 >
    <cfset foundservices = 0>
    <cfset foundpaint = 0>
    <cfset foundcarpet = 0>
    <cfset foundall = 0>

    <cfset resumefabric = 1>
    <cfset resumeservices = 1>
    <cfset resumepaint = 1>
    <cfset resumecarpet = 1>

    <!-- now work through the schedule to find stuff for the week
        in question for each column -->
    <cfloop index="j" from ="1" to= "#ArrayLen(buildstart)#">

    <!-- does the activity start this week ? -->
    <cfif wk[i] eq buildstart[j]>
        <cfif allactivz[j] eq 1>
            <cfset foundall = 1>
            <cfset numweeksalt = buildend[j] - buildstart[j] + 1>
            <cfset activityall = activityz[j]>
        <cfelse>
            <cfif fabricz[j] eq 1>
                <cfset foundfabric = 1>
                <cfset numweeksfabric = buildend[j] - buildstart[j] + 1>
                <cfset activityfabric = activityz[j]>
            <cfelse>
                <cfif servicesz[j] eq 1>
                    <cfset foundservices = 1>
                    <cfset numweeksservices = buildend[j] - buildstart[j] + 1>
                    <cfset activityservices = activityz[j]>
                <cfelse>
                    <cfif paintz[j] eq 1>
                        <cfset foundpaint = 1>
                        <cfset numweekspaint = buildend[j] - buildstart[j] +
1>
                        <cfset activitypaint = activityz[j]>
                    <cfelse>

```

```

+ 1>
                                <cfif carpetz[j] eq 1>
                                <cfset foundcarpet = 1>
                                <cfset numweekscarpet = buildend[j] - buildstart[j]

                                <cfset activitycarpet = activityz[j]>
                                </cfif>
                            </cfif>
                        </cfif>
                    </cfif>
                </cfif>
            </cfif>
        </cfif>
    </cfif>
</cfloop>

<!-- display all across all columns -->
<cfif foundall eq 1>
    <td rowspan=#numweeksfabric# colspan=4 valign="top">
        #activityall#
    </td>
<cfelse>

<!-- display fabric in first column - working out
how many rows it should span -->
<cfif foundfabric eq 1>
    <td rowspan=#numweeksfabric# valign="top">
        #activityfabric#
    </td>
    <!-- make sure we don't generate anything till this one
        is finished -->
    <cfset resumefabric = i+#numweeksfabric#>
<cfelse>
    <cfif resumefabric gte i>
        <td></td>
    </cfif>
</cfif>

<!-- display services in first column - working out
how many rows it should span -->
<cfif foundservices eq 1>
    <td rowspan=#numweeksservices# valign="top">
        #activityservices#
    </td>
    <cfset resumeservices = i+#numweeksservices#>
<cfelse>
    <cfif resumeservices gte i>
        <td></td>
    </cfif>
</cfif>

<!-- display paint in first column - working out
how many rows it should span -->
<cfif foundpaint eq 1>
    <td rowspan=#numweekspaint# valign="top">
        #activitypaint#
    </td>
    <cfset resumepaint = i+#numweekspaint#>
<cfelse>
    <cfif resumepaint gte i>
        <td></td>
    </cfif>
</cfif>

```

```

        </cfif>
    </cfif>

    <!-- display carpet in first column - working out
         how many rows it should span -->
    <cfif foundcarpet eq 1>
        <td rowspan=#numweekscarpet# valign="top">
            #activitycarpet#
        </td>
        <cfset resumecarpet = i+#numweekscarpet#>
    <cfelse>
        <cfif resumecarpet gte i>
            <td></td>
        </cfif>
    </cfif>

    </cfif>

</cfif>

</tr>
</cfloop>

</table>

</cfif>

```

<p></p>

To keep things simple the work to be carried out has been grouped into three main work packages. And they are as follows

<p></p>

<a name="fabric"><h2>Fabric Work</h2></a><br>

The first package is the work that it is to be carried by our Principal Contractor Messers McCarthy Joiners. McCarthy Joiners will be doing all the building and fabric work needed. In other words, they will be working on the shell of the building - on walls, floors, ceilings, entrances, ramps etc. They will start work on the 2nd May. The demolition work needed will be done first. The new Sanctuary floor will be one of the last jobs done, as this will avoid damaging the floor.

<p>

This work, allowing for some tidying up, should take some 13 weeks and on this basis would be completed at the end of July.

<p></p>

<a name="services"><h2>Services</h2></a><br>

The second package is the work needed to install all the services systems needed and fixtures such as kitchen units. The work includes putting in electrical, heating, plumbing and ventilation systems. It will be carried out by various sub-contractors (Joe Hambly will be doing the heating and plumbing work) and will take place concurrently with, or directly following on, the building and fabric work.

<p></p>

Fixtures such as the kitchen will be done last, i.e. in late July or early August.

<p></p>

<h2><a name="finishing">Finishing work</a></h2>

The third package can be split into 2 stages.

<p></p>

<a name="paint">a</a>. The first stage takes in any painting and decorating needed. This stage

starts in early August and could take up to 4 weeks.

<p></p>

<a name="carpet">b</a>. The final stage, estimated as taking 3 weeks, involves laying down carpets

and floor covers and doing any final outfitting needed, such as returning items to the Church, installing any new furniture bought such as the new sanctuary chairs. The radiators will be re-installed and tested before any carpets are laid down.

<p></p>

On the basis of the above schedule the Church would be available for services in mid to late September.

<!-- allow the building curators to edit the schedule -->

<cfif check.count eq 1 and session.admin eq 1>

<hr>

<h2>ADMIN</h2>

<form action="addbuilding.cfm" method="post">

<textarea name="newbuilding" cols=80 rows=5></textarea><br>

<input type="submit" name="go" value=" Add Building Announcement ">

</form>

<a href="adminbuild.cfm">Admin (add new scheduled activities or edit them)</a>

</cfif>

<p>

</td>

</tr>

</table>

</body>

</html>

</cfoutput>

## Page E

```
<!-- If session has timed out, start again --->
<cfif not IsDefined("session.id")>
    <cflocation url="index.cfm" addToken="No">
</cfif>

<cfif not IsDefined("Cookie.userid")>
    <cfcookie name = "userid"
        value = "#session.id#"
        expires = "30">
</cfif>

<cftry>

    <CFQUERY NAME="test" DATASOURCE="xxxxx" >
        SELECT L4USER.ID, email, name, surname, nickname
        FROM L4USER NATURAL JOIN L4PERSON
        WHERE L4PERSON.id = #session.id#

    </cfquery>
    <cfif not IsDefined("session.Username")> <cfset session.Username =
test.email></cfif>
    <cfif not IsDefined("session.name")> <cfset session.name = test.name></cfif>
    <cfif not IsDefined("session.surname")> <cfset session.surname =
test.surname></cfif>
    <cfif not IsDefined("session.nickname")> <cfset session.nickname =
test.nickname></cfif>
    <cfif not IsDefined("session.admin")>
        <cfif #checkdeacon.count# eq 1> <cfset session.admin=1>
        <cfelse> <cfset session.admin = 0>
        </cfif>
    </cfif>
    <cfif not IsDefined("session.adminMessage")> <cfset session.adminMessage = '
'></cfif>

<cfquery name="checkdeacon" datasource="xxxxx">
    select count(*) as count from deacon where id = '#session.id#'
</cfquery>

<cfset session.thispage = "notice board">

<cfinclude template="log.cfm">

<cfquery name="getpages" datasource="xxxxx">
    select * from pages where pagename='#session.thispage#'
</cfquery>
<cfset session.thisPageURL=#getpages.url#>
<cfset session.thisPageHeading = #getpages.pageheading#>

<!-- put them into arrays for display -->

<cfquery name="getmens" datasource="xxxxx">
    select * from activities where pulpit like "Mens Fellowship%" order by year, month, day,
number
</cfquery>
<cfset mendsays = ArrayNew(1)>
```

```

<cfset menmonths = ArrayNew(1)>
<cfset menyears = ArrayNew(1)>
<cfset menname = ArrayNew(1)>
<cfset mentime = ArrayNew(1)>
<cfset i=0>
<cfloop query="getmens">

    <CFSET newdate = #CREATEODBCDATE("#month##day##year#")#>

    <cfif #newdate# gte #today#>
        <cfset i = i+1>
        <cfset mendsays[i] = day>
        <cfset menmonths[i] = month>
        <cfset menyears[i] = year>
        <cfset menname[i] = pulpit>
        <cfset mentime[i] = time>
    <cfbreak>
    </cfif>
</cfloop>

<cfquery name="getthought" datasource="xxxxx">
select thethought from thought
</cfquery>
<cfquery name="getDeadline" datasource="xxxxx">
select day,month,year from deadline
</cfquery>
<!-- now get special announcements -->
<cfquery name="getann" datasource="xxxxx">
select * from announce order by number desc
</cfquery>

<!-- check if notices are emailed -->
<cfquery name="getmailing" datasource="xxxxx">
select count(*) as count from mailint where userid=#session.id#
</cfquery>

<cfset getsMail = #getmailing.count#>

<cfset announce = ArrayNew(1)>

<cfset i = 1>
<cfloop query="getann">
    <CFSET newdate = #CREATEODBCDATE("#month##day##year#")#>
    <!-- relevant announcement? -->
    <cfif #newdate# gte #today#>

        <cfset announce[i] = heading>
        <cfset i = i + 1>
    </cfif>
</cfloop>

<cfquery name="meetings" datasource="xxxxx">
select * from ladymeetings order by year, month, day, number
</cfquery>
<cfquery name="diary" datasource="xxxxx">
select * from ladydiary order by year, month, day, number

```



```

</cfquery>
<!-- Now put the duties into arrays for display -->
<cfset meetdays = ArrayNew(1)>
<cfset meetmonths = ArrayNew(1)>
<cfset meetyears = ArrayNew(1)>
<cfset meetdates = ArrayNew(1)>
<cfset meetname = ArrayNew(1)>

<cfset i = 1>

<cfloop query="meetings">
    <CFSET meetdates[i] = #CREATEODBCDATE("#month##day##year#")#>

    <cfif #meetdates[i]# gte #today#>

        <cfset meetdays[i] = day>
        <cfset meetmonths[i] = month>
        <cfset meetyears[i] = year>
        <cfset meetname[i] = activity>
        <cfset meettime[i] = time>
        <cfset i = i + 1>

    </cfif>
</cfloop>

<cfquery name="getall" datasource="xxxxx">
    select * from activities order by year, month, day, number
</cfquery>

<cfset actdays = ArrayNew(1)>
<cfset actmonths = ArrayNew(1)>
<cfset actyears = ArrayNew(1)>
<cfset actpulpits = ArrayNew(1)>
<cfset acttimes = ArrayNew(1)>
<cfset actdates = ArrayNew(1)>

<cfset i = 1>
<cfloop query="getall">
    <CFSET newdate = #CREATEODBCDATE("#month##day##year#")#>

    <!-- relevant announcement? -->
    <cfif #newdate# gte #today#>

        <cfset actdays[i] = day>
        <cfset actmonths[i] = month>
        <cfset actyears[i] = year>
        <cfset actpulpits[i] = pulpit>
        <cfset acttimes[i] = time>
        <cfset actdates[i] = newdate>

        <cfset i = i + 1>

    </cfif>
</cfloop>

<cfquery name="getprayer" datasource="xxxxx">
    select * from extraPrayer order by number desc
</cfquery>
<cfquery name="getweekprayer" datasource="xxxxx">
    select * from prayer
</cfquery>

```

```

<cfoutput>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <meta name="keywords" content="" />
    <meta name="description" content="" />
    <title>Helensburgh Baptist Church</title>

    <script type="text/javascript"></script>

    <style type="text/css" media="screen">
        @import "zen2.css";
    </style>
</head>

<body>
<div id="container">
<cfinclude template="includelinks2.cfm">

    <div id="introgap"><br></div>

    <div id="intro">
        <cfif isdefined("session.completionMessage")>
<cfif session.completionMessage neq "">

<table ><tr><td width=100% bgcolor="##99CCFF">
<b><large>#session.completionMessage#</large></b>
</td></tr></table>
<cfset session.completionMessage = "">
</cfif></cfif>

<br align="left">

<table width="100%" class="notices">
<tr>
    <td width="33%" class="notices">
<cfif getprayer.RecordCount gt 0>
        <h2 class="notice">Special Prayer Requests</h2>
        <cfloop query="getprayer">
            <br>#getprayer.request#<br>
        </cfloop>
        </cfif>

        <cfset pos = RandRange(1,11)>

        <h2 class="notice">Pray For:</h2>
        <cfswitch expression="#pos#" >
            <cfcase value="1">
                #getweekprayer.name1#
            </cfcase>
            <cfcase value="2">
                #getweekprayer.name2#
            </cfcase>
            <cfcase value="3">
                #getweekprayer.name3#

```

```

</cfcase>
<cfcase value="4">
#getweekprayer.name4#
</cfcase>
<cfcase value="5">
#getweekprayer.name5#
</cfcase>
<cfcase value="6">
#getweekprayer.name6#
</cfcase>
<cfcase value="7">
#getweekprayer.name7#
</cfcase>
<cfcase value="8">
#getweekprayer.name8#
</cfcase>
<cfcase value="9">
#getweekprayer.name9#
</cfcase>
<cfcase value="10">
#getweekprayer.name10#
</cfcase>
<cfcase value="11">
#getweekprayer.name11#
</cfcase>
</cfswitch>
<br>
<br><a href="prayer.cfm">Go to Prayer Calendar</a>
<br><br>
<h2 class="notice">T+U</h2> 
</td>
<td width="33%" class="notices">
<h2 class="notice">Breaking News</h2>

<cfquery name="getbreak" datasource="xxxxx">
select * from breaking order by number desc
</cfquery>
<cfloop query="getbreak">
<h3 class="verse">#heading#</h3>
#news#

<cfif session.admin eq 1>

<table border=1><tr>
<tr> <td> Change
<form action="editbreak.cfm" method="post" >
<input type="hidden" name="editnum" value="#number#">
<input type="image" src="edit.gif" name="Edit" value="Edit">
</form>
</td><td> Delete
<form
action="remove.cfm?table=breaking&remove=#number#&type=breaking
news&forwardpage=notices.cfm" method="post">
<input src="delete-icon.gif" type="image" name="RemoveActiv"
value="Remove">

</form>
</td>

```

```

        <td><form action="moveToNews.cfm" method="post">
            Move to News
            <input type="hidden" name="mv" value="#number#">
            <input src="move.png" type="image" name="Move" value="Move">
        </form>
    </td>
</tr>
</table>
</cfif>

</cfloop>

<cfif session.admin eq 1>
<form method="post" action="addbreak.cfm">
<h3 class="verse">Add Breaking News</h3>
Heading: <input type="text" name="heading" size="40"><br>
News:<br> <textarea name="news" cols=40 rows=10></textarea><br>
        <input type="submit" name="go" value="Add">
</form>
</cfif>

<h2 class="notice">News</h2>

<cfloop index="i" from="1" to= "#ArrayLen(announce)#">

    <cfif announce[i] neq "">
        #announce[i]#<br>
    </cfif>
</cfloop>
<br><a href="main.cfm">Read the News</a></td>

<cfset today = #Now()#>
<cfset todayDay = #Day(today)#>
<cfset todayMonth = #Month(today)#>
<cfset todayYear = #Year(today)#>
<cfset theDate = #CREATEODBCDATE("#todayMonth##todayDay##todayYear#")#>

<!-- find the next sunday -->
<cfloop index="i" from="1" to= "8">
    <cfset DayOfWeek = DateFormat(theDate,"dddd")>
    <cfif DayOfWeek eq 'Sunday'>
        <cfbreak>
    </cfif>
    <cfset theDate = #DateAdd('D',1,#theDate)#>
</cfloop>
<cfset sundayDay = #Day(theDate)#>
<cfset sundayMonth = #Month(theDate)#>
<cfset sundayYear = #Year(theDate)#>
<!-- check after noon -->
<cfset hour = #Hour(theDate)#>

<cfset count=0>
<cfset theDate = #CREATEODBCDATE("#todayMonth##todayDay##todayYear#")#>
<cfloop index="i" from="1" to= "8">
    <cfset DayOfWeek = DateFormat(theDate,"dddd")>
    <cfif DayOfWeek eq 'Friday'>

```

```

        <cfif hour le 12><cfbreak><cfelse><cfif count eq 0><cfset
count=1><cfelse><cfbreak></cfif></cfif>
    </cfif>
    <cfset theDate = #DateAdd('D',1,#theDate#)#>
</cfloop>
<cfset fridayDay = #Day(theDate)#>
<cfset fridayMonth = #Month(theDate)#>

<cfquery name="getrota" datasource="xxxxx">
    select * from sounddesk where day=#sundayDay# and month=#sundayMonth# and
year=#sundayYear#
</cfquery>

<cfquery name="getduties" datasource="xxxxx">
    select * from duties where day=#sundayDay# and month=#sundayMonth# and year=#sundayYear#
</cfquery>

<td width="33%" class="notices">
<!--<h2 class="notice">Deadline for Intimations</h2>
<cfset fridMonth = Month(theDate)>
12 noon, Friday: #fridayDay#/#fridMonth#
<p></p>-->

<cfif getsMail eq 1>

You are on the Intimation mailing list.
    <a href="dontMailToMe.cfm">(Click here to be removed from the list)</a>
<cfelse>


    <a href="mailToMe.cfm">I'd like be in the intimations mailing list</a>
</cfif>

<h2 class="notice">Sunday Sound Desk</h2>

<cfif getrota.RecordCount gt 0>

<cfset posspace = find(" ",#getrota.morning#)>
<cfset fname=Mid(#getrota.morning#,1,posspace-1)>
<cfset sname=Mid(#getrota.morning#,posspace+1,10)>

<b>Morning:</b> #getrota.morning# <a
href="getmail.cfm?name=#fname#&surname=#sname#&subject=Sound Desk"></a><br>
<cfset posspace = find(" ",#getrota.evening#)>
<cfset fname=Mid(#getrota.evening#,1,posspace-1)>
<cfset sname=Mid(#getrota.evening#,posspace+1,10)>
<b>Evening:</b> #getrota.evening# <a
href="getmail.cfm?name=#fname#&surname=#sname#&subject=Sound Desk"></a><br>
<cfelse>
No one assigned yet.
</cfif>

```

```

<cfif getduties.RecordCount gt 0>
<h2 class="notice">Sunday Deacons</h2>
<cfif #getduties.deacon3# eq "-">
#getduties.deacon1# & #getduties.deacon2#<br>
<cfelse>
<b>Morning:</b> #getduties.deacon1# & #getduties.deacon2#<br>
<b>Evening:</b> #getduties.deacon3# & #getduties.deacon4#
</cfif>
</cfif>

</td>

</tr>
<tr>

<cfset pos = 1>
<cfquery name="getnotice" datasource="xxxxx">
    select * from notice
</cfquery>
<cfloop query="getnotice">
<td width="33%" class="notices">
<h2 class="notice">#heading#</h2>
#notice#

<cfif checkdeacon.count eq 1 and session.admin eq 1>
<br>
<!-- <table border=0><tr><td>-->
<form action="editnotice.cfm" method="post" >
        <input type="hidden" name="edit" value="#number#">
        <input type="image" src="edit.gif" name="EditActiv" value="Change">
        </form>
        <!-- </td><td>-->
<form action="removenotice.cfm" method="post">
<input type="hidden" name="remove" value="#number#">
<input src="delete-icon.gif" type="image" name="RemoveActiv" value="Remove">
</form><!-- </td><td>-->
<a href="mailnotice.cfm?noticeNum=#number#">Email to Members</a>
<!-- </td></tr></table>-->
</cfif>
</td>
<cfset pos = pos+1>
<cfif pos gt 3>
</tr><tr>
<cfset pos = 1>
</cfif>
</cfloop>

<td width="33%" class="notices">
<h2 class="notice">Activities</h2>
        <cfset displayed=0><cfset displayedtom=0>
        <cfloop index="i" from ="1" to= "#ArrayLen(actdays)#">

                <cfif #actdates[i]# eq #today#>

```

```

                                <cfif displayed eq 0>
                                    <h3 class="notice"> Today's
Activities</h3>
                                </cfif>
                                <cfset displayed=1>
                                <b>#acttimes[i]#:</b> #actpulpits[i]#
                                </cfif>
                                <cfif actdates[i] gt today>
                                    <cfbreak>
                                </cfif>
                                </cfloop>

                                <cfloop index="i" from ="1" to= "#ArrayLen(actdays)#">
                                    <cfif #actdates[i]# eq #today#+1>
                                        <cfif displayedtom eq 0>
                                            <h3 class="notice"> Tomorrow's
Activities</h3>
                                        </cfif>
                                        <cfset displayedtom=1>
                                        <b>#acttimes[i]#:</b> #actpulpits[i]#
                                        </cfif>
                                        <cfif #actdates[i]# gt #today#+1>
                                            <cfbreak>
                                        </cfif>
                                        </cfloop>

                                <cfif displayed eq 0 and displayedtom eq 0>
                                <h3 class="notice">Next Activity</h3>
                                #actdays[1]# #MonthAsString(actmonths[1])#: #actpulpits[1]#
                                </cfif>
                                <p></p><a href="activities.cfm">Go to Activities</a>

</td>

<cfset pos = pos+1>
<cfif pos gt 3>
</tr><tr>
<cfset pos =1>
</cfif>

<td width="33%" class="notices">
    <h2 class="notice">Next Ladies' Meeting</h2>
    <cfif #ArrayLen(meetdays)# eq 0>
        -
    <cfelse>
        #meetdays[1]# #MonthAsString(meetmonths[1])#, #meettime[1]#: #meetname[1]#
    </cfif>
    <br><br><a href="ladies.cfm">Go to Ladies Page</a>

</td>

<cfset pos = pos+1>
<cfif pos gt 3>
</tr><tr>
<cfset pos =1>
</cfif>

```

```

<cfif #ArrayLen(mendays)# gt 0>
<td width="33%" class="notices">
    <h2 class="notice">Next Mens' Meeting</h2>

    #mendays[1]# #MonthAsString(menmonths[1])#, #mentime[1]#: #menname[1]#

    <br><br><a href="http://www.hbchurch.co.uk/mens.php">Go to Men's
Page</a>
</td>    <cfset pos = pos+1>
    </cfif>

<cfif pos gt 3>
</tr><tr>
<cfset pos = 1>
</cfif>

<td width="33%" class="notices">
    <h2 class="notice">Thought of the Month</h2>
    #getthought.thethought#
    <br><br><a href="prayer.cfm">Go to Prayer Calendar</a>
</td>

<cfset pos = pos+1>
<cfif pos gt 3>
</tr><tr>
<cfset pos = 1>
</cfif>

<td width="33%" class="vacant" >
<br><br>
<cfif checkdeacon.count eq 1>
<a href="getnotice.cfm">Add Notice</a>
<cfelse>
Advertise here!
</cfif>
</td>
</tr>
</table>

<cfinclude template="footer.cfm">

</div>

<div id="introgap"><br></div>

</body>
</html>

</cfoutput>
<cfcatch>
<cfoutput>
<h1> Sorry, but there is a major problem with the system. Please try again tomorrow</h1>
</cfoutput>
</cfcatch>
</cftry>

```